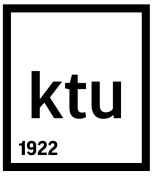


KAUNAS UNIVERSITY OF TECHNOLOGY
FACULTY OF MATHEMATICS AND NATURAL SCIENCES

Smoothed Particle Hydrodynamics Method for Gas Mixture Detonation Modelling

Bachelor's Final Degree Project

Michail Kiričkov
Project Author

Lekt. Dr. Rasa Šmidtaitė
Supervisor (applied mathematics programme)

Doc. Dr. Vytautas Stankus
Supervisor (applied physics programme)

Kaunas
2020

KAUNAS UNIVERSITY OF TECHNOLOGY
FACULTY OF MATHEMATICS AND NATURAL SCIENCES

Smoothed Particle Hydrodynamics Method for Gas Mixture Detonation Modelling

Bachelor's Final Degree Project

Applied mathematics
(code 612G10002) major study programme

Lecturer Dr. Rasa Šmidtaitė
Supervisor

Doc. Dr. Liepa Bikulčienė
Reviewer

Applied physics
(code B4026M21) minor study programme

Doc. Dr. Vytautas Stankus
Supervisor

Doc. Dr. Aleksandras Iljinas
Reviewer

Kaunas
2020

KAUNAS UNIVERSITY OF TECHNOLOGY
FACULTY OF MATHEMATICS AND NATURAL SCIENCES
Kiričkov Michail

Smoothed Particle Hydrodynamics Method for Gas Mixture Detonation Modelling

Declaration of Academic Integrity

I confirm that the final project of mine, Kiričkov Michail , on the topic „Smoothed Particle Hydrodynamics Method for Gas Mixture Detonation Modelling“ is written completely by myself; all the provided data and research results are correct and have been obtained honestly. None of the parts of this thesis have been plagiarised from any printed, Internet-based or otherwise recorded sources. All direct and indirect quotations from external resources are indicated in the list of references. No monetary funds (unless required by Law) have been paid to anyone for any contribution to this project. I fully and completely understand that any discovery of any manifestations/case/facts of dishonesty inevitably results in me incurring a penalty according to the procedure(s) effective at Kaunas University of Technology.

Michail Kiričkov. Smoothed Particle Hydrodynamics Method for Gas Mixture Detonation Modelling. Bachelor's Final Degree Project

Supervisor (major study programme) lecturer dr. Rasa Šmidtaitė, Faculty of Mathematics and Natural Sciences, Department of Applied Mathematics, Kaunas University of Technology.

Study field and area (study field group): Applied Mathematics.
Keywords: numerical methods, computational science, modelling , computational fluid dynamics.

Supervisor (minor study programme) doc. dr. Vytautas Stankus, Faculty of Mathematics and Natural Sciences, Department of Physics, Kaunas University of Technology.

Study field and area (study field group): Applied Physics.
Keywords: hydrodinamics, detonation, explosion modelling, compressible gas.

Kaunas, 2020. 50 pages.

Summary

Numerical method, based on the classical scheme of Smoothed Particle Hydrodynamics (SPH) is presented. SPH method was extended with combustion model, for detonation modelling. For detonation modelling have been used Euler equations for gas motion and mass transportation equation for mixture components. Meshless smoothed particle hydrodynamics metodology is apllied for governing equations discretization. Reaction velocity is defined by Arrenius law. Simplified chemical reaction kinetics is considered. In such setting this task has been solved for the first time in the world.

Michail Kiričkov. Suglodintų dalelių hidrodinamikos metodo taikymas dujų mišinio detonacijos modeliavimui. Bakalauro baigiamasis projektas.

Vadovė dr. Rasa Šmidtaitė; Kauno technologijos universitetas, Matematikos ir Gamtos mokslų fakultetas, Taikomosios matematikos katedra.

Studijų kryptis ir sritis (studijų krypčių grupė): Taikomoji matematika
Reikšminiai žodžiai: skaitinė hidrodynamika, skaitinis modeliavimas, skaitiniai metodai.

Vadovas doc. dr. Vytautas Stankus; Kauno technologijos universitetas, Matematikos ir Gamtos mokslų fakultetas, Fizikos katedra.

Studijų kryptis ir sritis (studijų krypčių grupė): Taikomoji fizika.
Reikšminiai žodžiai: sprogimas, detonacija, suspaudžiama srovė, dujos, cheminė kinetika.

Kaunas, 2020. 50 puslapiu.

Santrauka

Detonavimo modeliavimas naudojant Eulero lygtis dujų judesio aprašymui ir masės transportavimo lygtys mišinio komponentams. Betinklelinė suglodintu dalelių hidrodinamikos metodologija yra naudojama valdymo lygčių diskretizavimui. Reakcijos greitis apibrėžiamas Arenio dėsniu. Nagrinėjama supaprastinta cheminių reakcijų kinetiką. Tokioje aplinkoje šis užduotis yra sprendžiamas pirma karta pasaulyje.

Contents

List of Figures

Density profiles for the shock tube problem obtained by us

List of Symbols

The next list describes several symbols that will be later used within the body of the document

β	reaction progress variable
δ	Dirac delta function
$\frac{D}{Dt}$	material derivative
γ	ratio of specific heats
ω	chemical reaction rate
ρ	density
a	speed of sound
c_p	specific heat at constant pressure
c_v	specific heat at constant volume
D	domain
D_{CJ}	Chapman-Jouguet detonation speed
E	total energy of the mixture per unit mass
e	specific internal energy
E_a	global activation energy
f	any variable defined on the spatial co-ordinates x
h	characteristic distance between the particles
K	pre-exponential constant
$L_{1/2}$	half-reaction zone length
M	Mach number
m	mass of particle
N	dd
P	product
p	pressure
Q	chemical energy release
R	dimensionless distance betwseen particles

R	reactant
R	universal gas constant
T	temperature
t	time
u	velocity in the x-direction
u_i	velocity in the x_i-direction
v	velocity in the y-direction
v_i	velocity of i-particle
W	smoothing kernel
w	velocity in the z-direction
x	coordinate
y	coordinate
z	coordinate

Acknowledgments

Firstly I would like to thank my first, superviser and guide in Computational Fluid Dynamics, Head of Aerodynamics Department in Saint-Petersburg State Technical University, professor DSc Eugeny M. Smirnov, for his support and inspiration over last 20 years. It was his idea to come to Smoothed Particle Hydrodynamics method from well studied Finite Volume Methods.

I would also like to thank Dr. Algis Džiugys, Chief Research Associate in Laboratory of combustion processes of Lithuanian Energy Institute, for his help and support and for keeping fire in my inspiration for CFD in Lithuania.

I wish to thank professor Dr. Antonio Souto-Iglesias from Universidad Politécnica de Madrid, for his hearted and kind wish to help me to go through Smoothed Particle Hydrodynamics. It was his idea to consider detonation phenomena.

I wish to thank Dr. Jorge Yáñez Escanciano from Karlsruher Institut für Technologie for his help and support in combustion modelling.

I thank my official superviser, Dr. Rasa Šmidtaitė in Kaunas University for Technology, for his guide and mathematical thinking.

I also wish to thank my second supervisor, Dr. Vytautas Stankus for his guide and supervision.

I would like to thank professor Dr. Gui-Rong Liu from Department of Aerospace Engineering and Engineering Mechanics, University of Cincinnati, and co-workers for providing their SPH source code.

And finally I wish to thank my doctor Agnė Elijošienė from Kaunas Hospital, because without her medical support this work at all would be impossible.

Also I appreciate the financial support from the State Study Fund of Republic of Lithuania.

So this work is a result of cooperation of very different people from different parts of the world.

1. Overview

1.1. Introduction

The theory of detonation is one of the most important fields of gas dynamics. Soon after detonation phenomena was discovered, it have got right explanation, based on theory of shock waves. According to it, gas heating in the shock wave initiate explosion reaction, which energy support propagation of detonation wave. Such way was developed hydrodynamic theory of detonation [3].

Although gaseous detonation waves have been studied extensively for many years, methods for study detonation phenomena becomes more and more sophisticated with development of Computational Fluids Dynamics (CFD). Density profiles for the shock tube problem obtained by us The most common approach modeling of the detonation initiation and propagation in supersonic combustible mixtures using the three dimensional Euler equations for a mixture of different thermally perfect species with chemically reactive source terms [4, 5, 6, 7].

Usually it was modeled with well-studied CFD methods - Finite Difference Method (FDM), Finite Volume Method (FVM), Volume of Fluid Method (VFM). These methods uses standard approach (Eulerian), when fluid quantities is defined on a spatial grid, computing derivatives using finite difference or finite volume schemes.

Recent years with development of powerful Graphics Processor Units (GPU), which allows to perform calculations using personal computers and its clusters, has been increased interest to Lagrangian methods in CFD [8]. This approach is alternative to grid methods, when fluid quantities are carried by a set of moving interpolation points which follow the fluid motion. One of Lagrangian methods is the Smoothed Particle Hydrodynamics (SPH) method. It was introduced by Lucy (1977) [9] and by Gingold and Monaghan (1977) [10].

However, although SPH method was developing such long time, and it has many advantages, its application to modeling of reactive flows and combustion processes is still highly limited. The first work concerning SPH application to the combustion processes has appeared in 2003 [11, 12].

As we can notice, since then, no work was performed for applying SPH to combustion modeling So it was decided to fill this gap in SPH developing.

Usually in grids methods combustion model is included by adding transport equation for combustion products and closing model for energy equation. The same approach in SPH method framework is applied in this work. Because of complexity of applying such task for viscous flows regimes, for modeling was chosen detonation regime process. In this case Navier-Stoces equations simplified to Euler equations, there is no need to consider diffusion and turbulence phenomenon.

Creating SPH method program code from zero is rather difficult and complicating task, therefore it was decided to incorporate combustion model into one of existing free SPH codes. Among numerous SPH codes [13], such as SPHYSICS [14], or Daniel Price's program - NDSPMHD [15, 16, 17] was chosen professor's Gui-Rong Liu SPH code, provided by GRLab-Computation for Sustainability [18, 19]. In comparison with D.Price code it is better documented, and it was important for developing, because we were limited in time.

1.2. Smoothed particle hydrodynamics for explosion modeling

The use of SPH based methods to model shock wave problems has been relatively sparse, both due to historical reasons, as the method was originally developed for studies of astrophysical gas dynamics [9, 10].

Afterwards, due to the advantages of SPH method in tracking moving interfaces and dealing with large deformation, it has been applied to solve fluid dynamic problems and beyond, explosion, and hypervelocity impact problems. Liu, Liu et al. [18, 20, 21, 22, 23] investigated the feasibility of using SPH to simulate explosion of high explosive, underwater explosion, shaped charge detonation, and so on.

Since then, SPH were widely allied for modeling blast and detonation problems [24, 25].

M.Omang et al. [26] applied Regularized Smoothed Particle Hydrodynamics (RSPH) to study propagation and reflection of blast waves from detonations of the high explosives C-4 and TNT.

G. Wang et al. applied SPH method for modeling ANFO (ammonium nitrate/fuel oil) detonation [27].

Gang Yang et al. used SPH method in simulating shock initiation process. [28]

Wang et al. [29] modeled detonation of aluminized explosives, using SPH and source term in equation of energy. However, they used special form equation of state for detonation products. We shall try to extent this approach and and shall use the equation of state for ideal gas both for reactants and reaction products.

All these cases concerned large scale modeling In this work, we will try to apply SPH method for modeling detonation on the scale of detonation front.

In all above mentioned works was used special form of equation of state for detonation products.

Usually in most works was used the standard Jones-Wilkins-Lee (JWL) equation, Dobratz [30].

$$\rho = A(1 - \frac{\omega\eta}{R_1})e^{-\frac{R_1}{\eta}} + B(1 - \frac{\omega\eta}{R_2})e^{-\frac{R_2}{\eta}} + \omega\eta\rho_0 e \tag{1}$$

In our work we will use usual form of equation of state. In dimensionless form equation of state has following form

$$p = (\gamma - 1)\rho e \tag{2}$$

1.3. Objective of this work

Main purpose of this work - to make first steps in Baltic region to create and develop methodology, combustion model and computer program for combustion modeling of chemically reactive flows, based on SPH method.

2. Mathematical model

2.1. Governing equations

Since the detonation and expansion speed are extremely high, the gaseous products can be assumed to be inviscid and the explosion process is adiabatic. Density profiles for the shock tube problem obtained by us The motion of a inviscid, heat conducting, reacting, and compressible medium is described by the continuity, momentum, energy, and species concentration equations. For a calorically perfect gas with an energy source for multi-component reaction system the conservation equations are [31, 32] (so-called time-dependent Euler equations)

$$\frac{\partial \rho}{\partial t} + \frac{\partial \rho u_i}{\partial x_i} = 0 \tag{3}$$

$$\frac{\partial \rho u_i}{\partial t} + \nabla(\rho u_i U) + \frac{\partial p}{\partial x_i} = 0 \tag{4}$$

$$\frac{\partial \rho e}{\partial t} + \nabla(\rho e u_i) = \beta Q \tag{5}$$

$$\frac{\partial \rho \beta}{\partial t} + \nabla(\rho \beta u_i) = \rho \Omega \tag{6}$$

where

$$e = \frac{p}{(\gamma - 1)\rho} + \frac{u^2}{2} \tag{7}$$

$$\Omega = -K(\beta - 1)\exp\left(\frac{-E_a}{T}\right) \tag{8}$$

and

$$T = \frac{p}{\rho} \tag{9}$$

where index notation with Einstein's summation convection is used for vectors and tensors in Cartesian coordinates $(i, j, k = 1,2,3)$ and x_i are the components of the position vector, u_i is the velocity vector components, ρ is the density, p is the pressure, β is the reaction progress variable, which varies between 1 (for unburned reactant) and 0 (for product), Ω is the reaction velocity, T is the temperature. Q and E_a represents non-dimensional heat release and activation energy.

Furthermore, in Cartesian coordinates, the vector ∇ is defined as

$$\nabla \equiv \overrightarrow{i}\frac{\partial}{\partial x} + \overrightarrow{j}\frac{\partial}{\partial y} + \overrightarrow{k}\frac{\partial}{\partial z} \tag{10}$$

In Lagrangian form eq. (1)-(4) can be rewritten as

$$\frac{D\rho}{Dt} = -\rho\nabla\cdot u \tag{11}$$

$$\frac{Du}{Dt} = -\frac{1}{\rho}\nabla p \tag{12}$$

$$\frac{De}{Dt} = -\frac{p}{\rho}\nabla u + \beta Q \tag{13}$$

$$\frac{D\rho\beta}{Dt} = -\rho\beta\nabla u + \rho\Omega \tag{14}$$

$$p = (\rho,e) \tag{15}$$

$$\frac{D}{Dt} \equiv \frac{\partial}{\partial t} + \left(\overrightarrow{V}\cdot\nabla\right) \tag{16}$$

where $\frac{D}{Dt}$ is called *substantial derivative* and in cartesian coordinates, has an expression

$$\frac{D}{Dt} = \frac{\partial}{\partial t} + u\frac{\partial}{\partial x} + v\frac{\partial}{\partial y} + w\frac{\partial}{\partial z} \tag{17}$$

Equation (8) represents a definition of the substatial derivative operator in vector notation; thus it is valid for any coordinate system.

The system of differential equations (1)-(4) is closed with the equation of state for an ideal gas

$$p = \rho RT \tag{18}$$

where R is a gas constant. In dimensionless form equation of state has following form

$$p = (\gamma - 1)\rho e \tag{19}$$

2.2. Dimensionless variables

When building numerical applications we usually prefer to remove physical units from equations and work with dimensionless variables. This simplifies the problem formulation and reduce computational round-off errors. Physical variables in previous equations are nondimensionalized (or scaled) with reference to the uniform unburned state ahead of the detonation front

The nondimensional variables are obtained from the physical variables as follows:

$$\rho^* = \frac{\rho}{\rho_0}, \quad u^* = \frac{u}{a_0}, \quad T^* = \frac{T}{\gamma T_0}, \quad p^* = \frac{p}{p_0} \tag{20}$$

Non-dimensional heat release and activation energy scaled as:

$$E_a^* = \frac{E_a}{RT_0}, \quad Q^* = \frac{Q}{RT_0} \tag{21}$$

where the superscript $*$ and the subscript 0 indicate the non-dimensional and the reference quantities, respectively.

The set of equations presented above include acoustic interactions and compressibility effects. Body forces (e.g. gravitational) and thermal radiation effects as well as heat effect due to viscous dissipation are neglected.

2.3. Simplified combustion Model

A simplified version of the detailed model is derived by adopting several assumptions. The first assumption is to consider only a single reaction in the model that is irreversible, such that the reactant goes to product, $R \longrightarrow P$, where R and P are the reactant and product, respectively. Secondly, the reactant and product have identical molecular masses,$\overline{M}_R = \overline{M}_P$. Therefore, the mixture molecular mass remains constant, $\overline{M} = \overline{M}_R = \overline{M}_P$, and thus, the mole and mass fractions are the identical. Thirdly, the Soret effects is neglected so thermal diffusion coefficients are set to zero, $D_R^T = D_P^T = 0$. Additionally, both the reactant and product are calorically perfect ideal gases and have identical specific heats, $c_{pR} = c_{pP} = c_p = c$, where c_p is the mixture specific heat at constant pressure and c a constant. Under these assumptions, the model reduces significantly. From here, the mass fraction of the reactant and product will be given by $1 - \beta$ and β, respectively.

3. Smoothed Particle Hydrodynamics

3.1. Interpolant and SPH methodology

The basis of the SPH approach is given as follows (Monaghan [33, 34], Liu, Liu [18]). We begin with the trivial identity

$$f(x) = \int f(x')\delta\left(|x - x'|\right) dx', \tag{22}$$

where f is any variable defined on the spatial co-ordinates x and δ refers to the Dirac delta function. This integral is then approximated by replacing the delta function with a smoothing kernel W with characteristic width h, such that

$$f(x) = \int f(x')W\left(|x - x'|,h\right) dx' + O(h^2) \tag{23}$$

Smoothing kernel W (or simply *kernel*) satisfy a number of conditions:

1. Normalization condition thet states

$$\int_\Omega W(x - x',h)dx' = 1 \tag{24}$$

2. Delta function property

$$\lim_{h\to 0} W(x - x',h) = \delta(x - x') \tag{25}$$

3. Compact condition

$$W(x - x',h) = 0, \text{ when } |x - x'| > \kappa h \tag{26}$$

   where κ is a constant related to the smoothing function for point at x, and defines effective (non-zero) area of smoothing length.

The integral interpolant (Eq. 21) is discretized onto a finite set of interpolation points (the particles) by replacing the integral by a summation and the mass element $\rho d\Omega$ with the particle mass m, i.e.

$$\langle f(x)\rangle = \int \frac{f(x')}{\rho(x')} W(x - x', h)\rho(x')dx' \approx \sum_{b=1}^{N_{\text{neigh}}} m_b \frac{f_b}{\rho_b} W(x - x_b, h) \tag{27}$$

Gradient terms may straightforwardly calculated by taking the derivative of (25), giving

$$\begin{aligned} \nabla f(x) &= \frac{\partial}{dx_i} \int \frac{f(x')}{\rho(x')} W(x - x',h)\rho(x')dx' + O(h^2) \\ &\approx \sum_b m_b \frac{f_b}{\rho_b} \nabla W(x - x_b,h) \end{aligned} \tag{28}$$

where

$$\nabla_a W_{ab} = \frac{x_a - x_b}{r_{ab}} \frac{\partial W_{ab}}{\partial r_{ab}} = \frac{x_{ab}}{r_{ab}} \frac{\partial W_{ab}}{\partial r_{ab}} \tag{29}$$

The problem is that using these expressions 'as is' in general leads to quite poor gradient estimates. Monaghan (1992) used more sophisticated approach of directly using equations (25) and (26). In this approach, the following two identities are introduced to place the dencity inside the gradient

operator,

$$\nabla \cdot f(x) = \frac{1}{\rho}[\nabla \cdot (\rho f(x)) - f(x) \cdot \nabla \rho] \tag{30}$$

$$\nabla \cdot f(x) = \rho[\nabla \cdot (\frac{f(x)}{\rho}) + \frac{f(x)}{\rho^2}) \cdot \nabla \rho] \tag{31}$$

these two identities are substituted into the integral in equation (26). Thus we get for particle a:

$$\nabla \cdot f(x_a) = \frac{1}{\rho_a}\left[\sum_{b=1}^{N_{\text{neigh}}} m_b[f(x_b) - f(x_a)] \cdot \nabla_a W_{ab}\right] \tag{32}$$

Density profiles for the shock tube problem obtained by us

$$\nabla \cdot f(x_a) = \rho_a\left[\sum_{b=1}^{N_{\text{neigh}}} m_b\big[(\frac{f(x_b)}{\rho_b^2}) - (\frac{f(x_a)}{\rho_a^2})\big] \cdot \nabla_a W_{ab}\right] \tag{33}$$

The advantage of these two equations is that the field function $f(x)$ appears in the form of paired particles.

3.2. Smoothing function

To be short, we omit all considerations of smoothing function construction. We refer to G. R. Liu and M. B. Liu book [18].

Here we will give only summary of smoothing functions, which G.R.Liu are using in his code.

They are defined for a particle at x, r is the distance between two points or particles at x and x', and R is defined as:

$$R = \frac{r}{h} = \frac{|x - x'|}{h}$$

1. Gaussian kernel by Gingold and Monaghan (1977)

$$W(R,h) = \alpha_d e^{-R^2} \tag{34}$$

   α_d is $5/4h$, $5/\pi h$, $105/16\pi h^3$ in one-, two- and three- dimensional space respectively

2. Cubic spline kernel by B-Spline by Monaghan and Lattanzio (1985)

$$W(R,h) = \alpha_d \times \begin{cases} \frac{2}{3} - R^2 + \frac{1}{2}R^3 & 0 \le R < 1 \\ \frac{1}{6}(2-R)^3 + \frac{1}{2}R^3 & 1 \le R < 2 \\ 0 & R \ge 2 \end{cases} \tag{35}$$

   α_d is $1/h$, $15/7\pi h^2$, $3/2\pi h^3$ in one-, two- and three- dimensional space respectively

3. Quintic kernel by Morris (1996)

$$W(R,h) = \alpha_d \times \begin{cases} (3-R)^5 - 6(2-R)^5 + 15(1-R)^5 & 0 \le R < 1 \\ (3-R)^5 - 6(2-R)^5 & 1 \le R < 2 \\ (3-R)^5 & 2 \le R < 3 \\ 0 & R \ge 3 \end{cases} \tag{36}$$

α_d is $1/120h$, $7/478\pi h^2$, $3/359\pi h^3$ in one-, two- and three- dimensional space respectively

3.3. Artificial viscosity

In oder to avoid unphysical oscillations around the shocked region, there is used artificial viscosity in momentum and energy equations. We follow Monahgan [33, 34], and Liu, Liu [18]). G.R.Liu in his code is using this formulation:

$$\Pi_{ij} = \begin{cases} \dfrac{-\alpha_{\Pi}\overline{c_{ij}}\phi_{ij} + \beta_{\Pi}\phi_{ij}^2}{\overline{\rho_{ij}}} & v_{ij} \cdot x_{ij} < 0 \\ 0 & v_{ij} \cdot x_{ij} \geq 0 \end{cases} \tag{37}$$

where

$$\phi_{ij} = \frac{h_{ij} v_{ij} \cdot x_{ij}}{|x_{ij}|^2 + \varphi^2} \tag{38}$$

$$\overline{c_{ij}} = \frac{1}{2}(c_i + c_j) \tag{39}$$

$$Densityprofilesfortheshocktubeproblemobtainedbyus \overline{\rho_{ij}} = \frac{1}{2}(\rho_i + \rho_j) \tag{40}$$

$$\overline{h_{ij}} = \frac{1}{2}(h_i + h_j) \tag{41}$$

$$v_{ij} = v_i - v_j, x_{ij} = x_i - x_j \tag{42}$$

Here α_{ϖ} , β_{ϖ} are constants that are all tyoically set around 1.0. The factor $\varphi = 0.1 h_{ij}$ is inserted to prevent numerical divergences when two particles are approacing each other. c and v represent the speed of sound and the particle velocity vector respectively.

3.4. Discretized equations

We follow G.R. Liu and M.B. Liu book [18], and give here only final formulation.

Continuity equation.

$$\frac{D\rho_i}{Dt} = \sum_{j=1}^{N_{\text{neighb}}} m_j (v_i - v_j) \cdot \nabla_i W_{ij} \tag{43}$$

Equation of motion.

$$\frac{Dv_i}{Dt} = -\sum_{j=1}^{N_{\text{neighb}}} m_j \left(\frac{p_i}{\rho_i^2} + \frac{p_i}{\rho_i^2} + \Pi_{ij}\right) \cdot \nabla_i W_{ij} \tag{44}$$

Energy equation.

Energy equation as given by G.R. Liu for using our model was unacceptable. By trial and error method was found expression for energy equation, for using with Arrenius law and one-step reaction rate variable. Derivation was mostly intuitive, because we had not time and preparation for deeper analysis.

If we simply add source term in form βQ, it leads to unphysical solution, because after detonation front $\beta = 1$. Therefore we decided to multiply dencity and specific heat of reaction by reaction velocity, it is changing from 0. to 1. and exist only in the reaction zone, in all other reaction velocity is equal to zero.

Sign of source term is positive, because we add heat of reaction.

So, finally we have:

$$\frac{De_i}{Dt} = \frac{1}{2}\sum_{j=1}^{N_{\text{neighb}}} m_j(\frac{p_i}{\rho_i^2} + \frac{p_i}{\rho_i^2} + \Pi_{ij})(v_i - v_j)\cdot\nabla_i W_{ij} + Q\rho_i\Omega_i \tag{45}$$

Equation for species concentration transport. Equation for spiecies transportation identical to continuity equation and derived using eq. (30) for velocity gradient.

$$\frac{D\beta_i\rho_i}{Dt} = \beta_i\sum_{j=1}^{N_{\text{neighb}}} m_j(v_i - v_j)\cdot\nabla_i W_{ij} + \rho_i\Omega_i \tag{46}$$

we divide eq. (44) by ρ_i and get

$$\frac{D\beta_i}{Dt} = \frac{\beta_i}{\rho_i}\sum_{j=1}^{N_{\text{neighb}}} m_j(v_i - v_j)\cdot\nabla_i W_{ij} + \Omega_i \tag{47}$$

However, this approuch was found unacceptable because of arising osscilations on the detonation front. Therefore we decided to use the second approach - like for enerrgy equation.

$$\frac{D\beta_i}{Dt} = -\sum_{j=1}^{N_{\text{neighb}}} m_j(\frac{\beta_i}{\rho_i^2} + \frac{\beta_j}{\rho_j^2})\cdot\nabla_i W_{ij}\cdot(v_j - v_i) + \Omega_i\cdot\rho_i \tag{48}$$

3.5. Timestepping. Leapfrog scheme

Equations of motions (41-44) then integrated numerically, using a popular leapfrog method. In the leapfrog method, the particle velocities and positions are offset by a half time step when when integrating the equations of motion. At the end of the first time step (t_0), the change in density, energy, velocity and reaction rate are used to evaluate density, energy, velocity and reaction rate at half a time step, while the particle positions are advanced in a full time step.

$$\begin{cases} t = t_0 + \Delta t \\ \rho_i(t_0 + \frac{\Delta t}{2}) = \rho_i(t_0) + \frac{\Delta t}{2} D\rho_i(t_0) \\ e_i(t_0 + \frac{\Delta t}{2}) = e_i(t_0) + \frac{\Delta t}{2} De_i(t_0) \\ v_i(t_0 + \frac{\Delta t}{2}) = v_i(t_0) + \frac{\Delta t}{2} Dv_i(t_0) \\ \beta_i(t_0 + \frac{\Delta t}{2}) = \beta_i(t_0) + \frac{\Delta t}{2} D\beta_i(t_0) \\ x_i(t_0 + \frac{\Delta t}{2}) = x_i(t_0) + \Delta t \cdot v_i(t_0 + \frac{\Delta t}{2}) \end{cases} \tag{49}$$

In order to keep the calculations consistent at each subsequent time step, at the start of each subsequent time step, the density, energy and velocity of each particle need to be predicted at half a time step to coincide the position:

$$\begin{cases} \rho_i(t) = \rho_i(t - \frac{\Delta t}{2}) + \frac{\Delta t}{2} D\rho_i(t - \Delta t) \\ e_i(t) = e_i(t - \frac{\Delta t}{2}) + \frac{\Delta t}{2} De_i(t - \Delta t) \\ v_i(t) = v_i(t - \frac{\Delta t}{2}) + \frac{\Delta t}{2} Dv_i(t - \Delta t) \\ \beta_i(t) = \beta_i(t - \frac{\Delta t}{2}) + \frac{\Delta t}{2} D\beta_i(t - \Delta t) \end{cases} \tag{50}$$

At the end of the subsequent time step, the particle density, internal energy, velocity, reaction rate and position are calculating in the standard leapfrog scheme

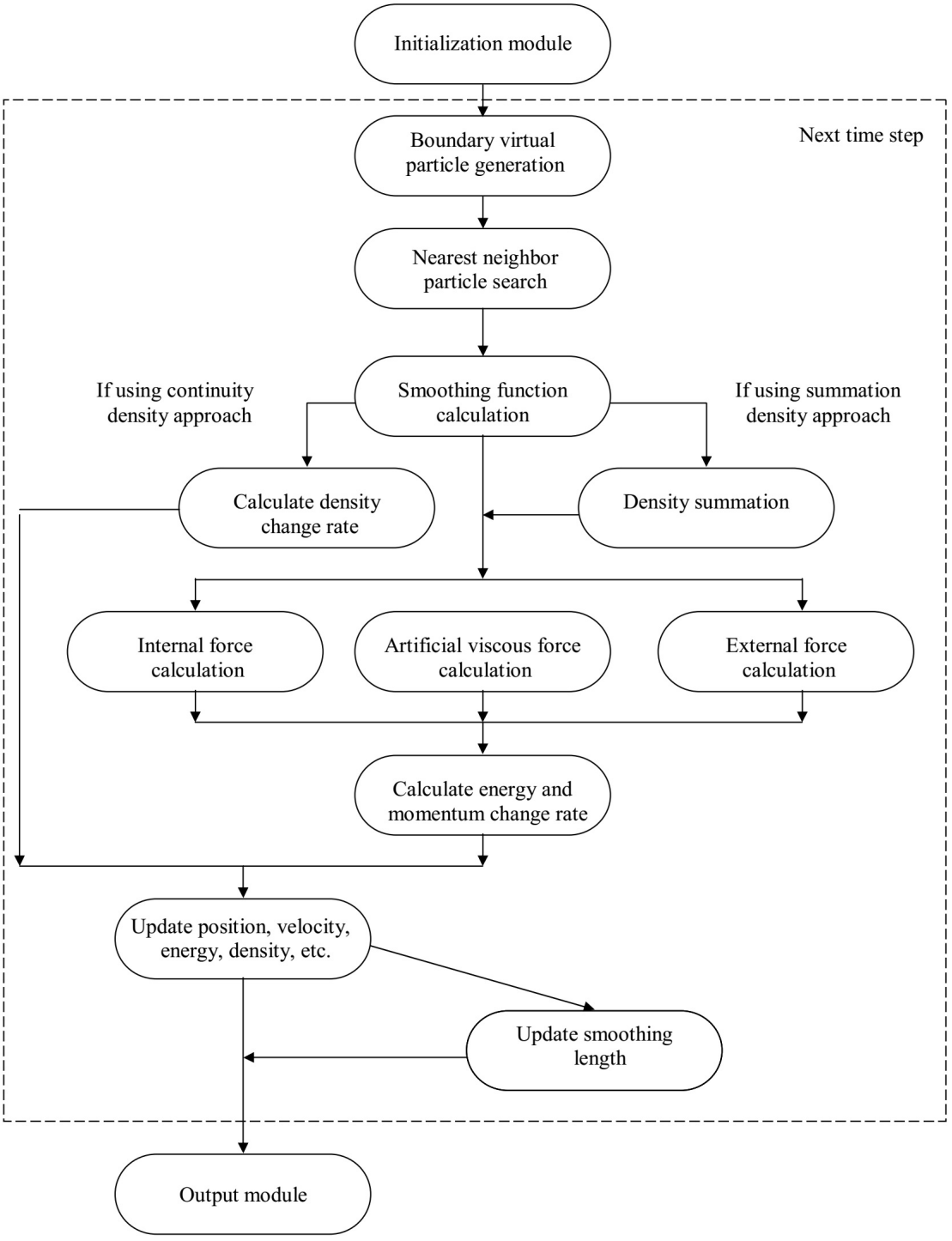
$$\begin{cases} t = t + \Delta t \\ \rho_i(t + \frac{\Delta t}{2}) = \rho_i(t - \frac{\Delta t}{2}) + \Delta t \cdot D\rho_i(t) \\ e_i(t + \frac{\Delta t}{2}) = e_i(t - \frac{\Delta t}{2}) + \Delta t \cdot De_i(t) \\ v_i(t + \frac{\Delta t}{2}) = v_i(t - \frac{\Delta t}{2}) + \Delta t \cdot Dv_i(t) \\ \beta_i(t + \frac{\Delta t}{2}) = \beta_i(t - \frac{\Delta t}{2}) + \Delta t \cdot D\beta_i(t) \\ x_i(t + \frac{\Delta t}{2}) = x_i(t) + \Delta t \cdot Dv_i(t + \frac{\Delta t}{2}) \end{cases} \tag{51}$$

The minimum timestep is defined by Courant-Friedrichs-Levy (CFL) condition. In this work, the time step is taken as [35].

$$\Delta t = \min\left(\frac{\xi h_i}{h_i \nabla \cdot v_i + c_i + 1.2(\alpha_{II} c_i + \beta_{II} h_i ! \nabla \cdot v_i |)}\right) \tag{52}$$

where ξ is the Courant number, taken around 0.3, α_{II} and β_{II} are the two coefficients in Monaghan type artificial viscosity. As recommended in [18] α_{II} is taken as $\alpha_{II} = 1$ and $\beta_{II} \approx 10...15$.

3.6. SPH algorithm



1 Figure Typical SPH algorithm block scheme

4. SPH program

4.1. Overview

The code is based on the SPH techniques presented in [18]. Treatments for the simulation of special phenomena (explosion, impact, penetration, etc.) are not listed in the code.

1. The code can be easily extended to other corrective or modified versions of SPH with proper treatment either on the kernel approximation or the particle approximation.
2. The code is implemented in three-dimensional space. Therefore, 1-, 2- and 3-dimensional simulations are possible.
3. The code can solve compressible flows with real viscosity. The shear stress and strain rate can all be calculated using the corresponding SPH approximations.
4. The code can be employed to simulate incompressible flows using the concept of artificial compressibility through a properly selected equation of state.
5. The code can be readily modified for application to hydrodynamics with material strength with a proper modification on the constitutive model and the equations of state.

4.2. ExactPack program packet

Los Alamos National Laboratory (USA) released software for code verification [36].

We used this software for getting exact solutions for our test problems.

ExactPack is a utility that integrates many of these exact solution codes into a common API (application program interface), and can be used as a stand-alone code or as a python package. ExactPack consists of python driver scripts that access a library of exact solutions written in Fortran or Python.

4.3. Our work

GRLab-Computation for Sustainability provides only basic core of SPH program. All features and components for solving concrete task should be added additionaly.

We added to the code:

1. module for solution initiation;
2. boundary conditions implementation as described in [37, 38]; we also found that wrong initiated solution tends to instability with negative reaction velocity, what causes nonphysical solution. There in boundary condition module every time step reaction rate variable was suppressed to 0. before detonation front, and 1. after it;
3. module for solution equation for reaction rate variable transport;
4. we modified time integration module, and added to it integration of reaction rate variable and addition of reaction heat in energy equation.

5. Adiabatic Sod shock tube test modelling

5.1. Problem parameters

The 1D Sod shock tube test [39] is frequently used for testing numerical codes, as there is an analytical solution available. With a membrane, the shock tube is divided into two chambers given the following initial conditions:

$$P = \begin{cases} 1.0 & x < 0. \\ 0.1 & x > 0. \end{cases} \tag{53}$$

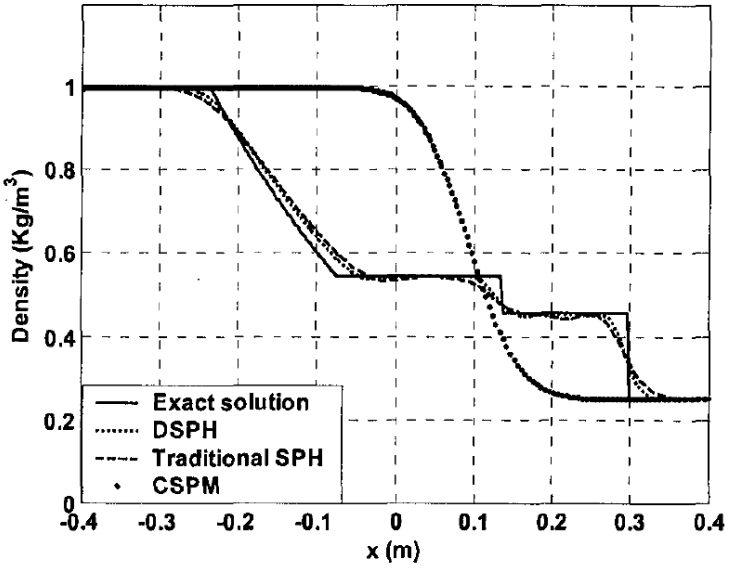
and

$$\rho = \begin{cases} 1.0 & x < 0. \\ 0.125 & x > 0. \end{cases} \tag{54}$$

where P and ρ are the pressure and density, respectively. The $\gamma = 1.4$ gas is assumed to be at rest initially. Assuming that the membrane is instantaneously removed, a shock is formed, moving into the low-pressure chamber, whereas a rarefaction wave moves into the high-pressure chamber.

5.2. G.R. Liu results

Here we reproduce results, obtained by G. R. Liu and published in his book [18].



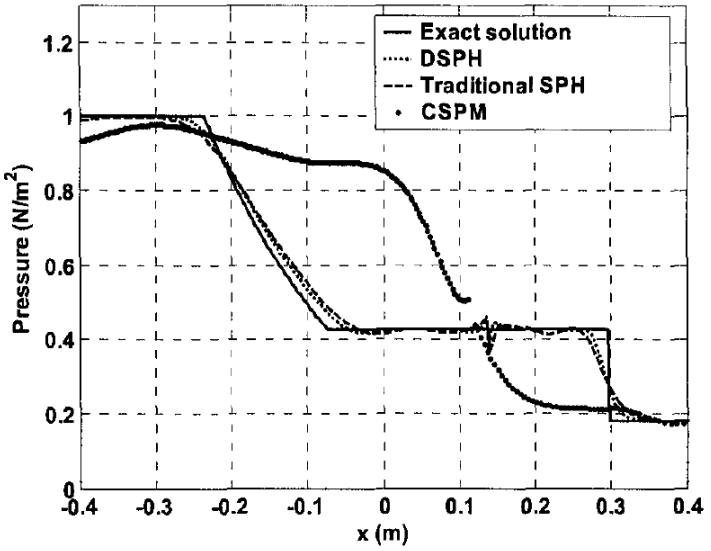
2 Figure Density profiles for the shock tube problem obtained by G. R. Liu (2003)

G. R. Liu used for his calculations different numerical schemes

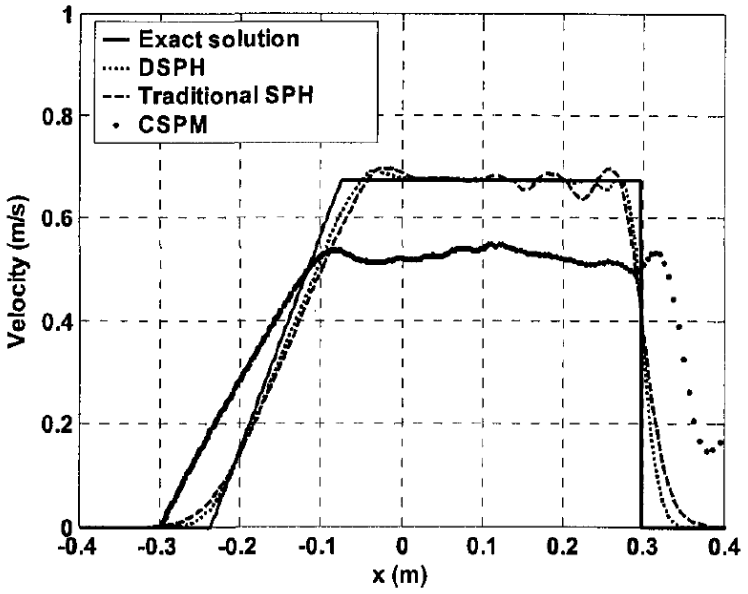
- Traditional SPH method
- Discontinuous Smoothed particle hydrodynamics (DSPH)
- Corrective Smoothed particle method (CSPM)

In G. R. Liu simulation the time step is 0.005 s using 400 particles.

As shown below, we have got much better results.



3 Figure Pressure profiles for the shock tube problem obtained by G. R. Liu (2003)



4 Figure Velocity profiles for the shock tube problem obtained by G. R. Liu (2003)

5.3. Results and discussion

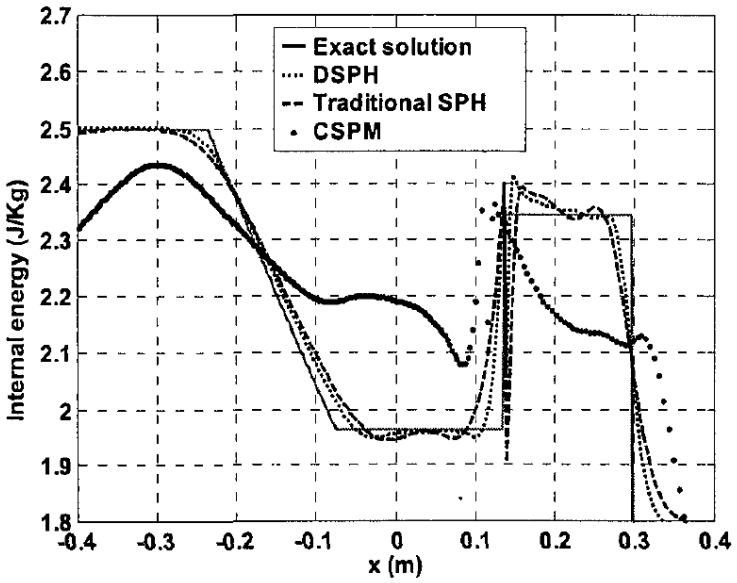
In the simulation, the time step is 0.00005 s and the simulation is carried out for 4000 steps respectively. A total of 4000, 8000 and 12000 particles are deployed in the one-dimensional problem domain. All particles have the same mass.

In the original G .R. Liu SPH code boundary conditions were not implemented. We added our modification to the code. but we did it in our way - we kept variables of 5 particles from the both sides with known boundary values and updated them every time step.

Figure 6, figure 7 , figure 8, figure 9 show, respectively, the density, pressure, velocity and internal energy profiles.

It can be seen that the obtained results agree well with the exact solution in the region $[-0.5, 0.5]$, The shock is observed at around $x = 0.25$.

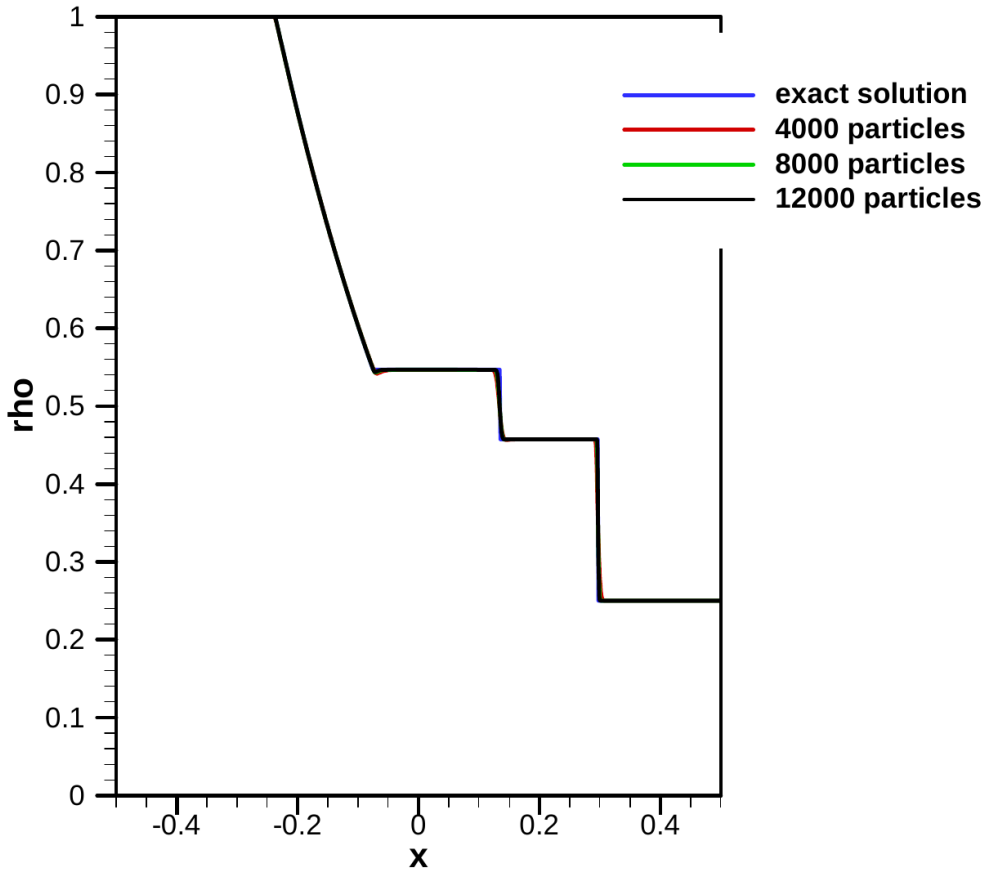
It can be seen however, using usual SPH method, unphysical oscilation in shock region appeared



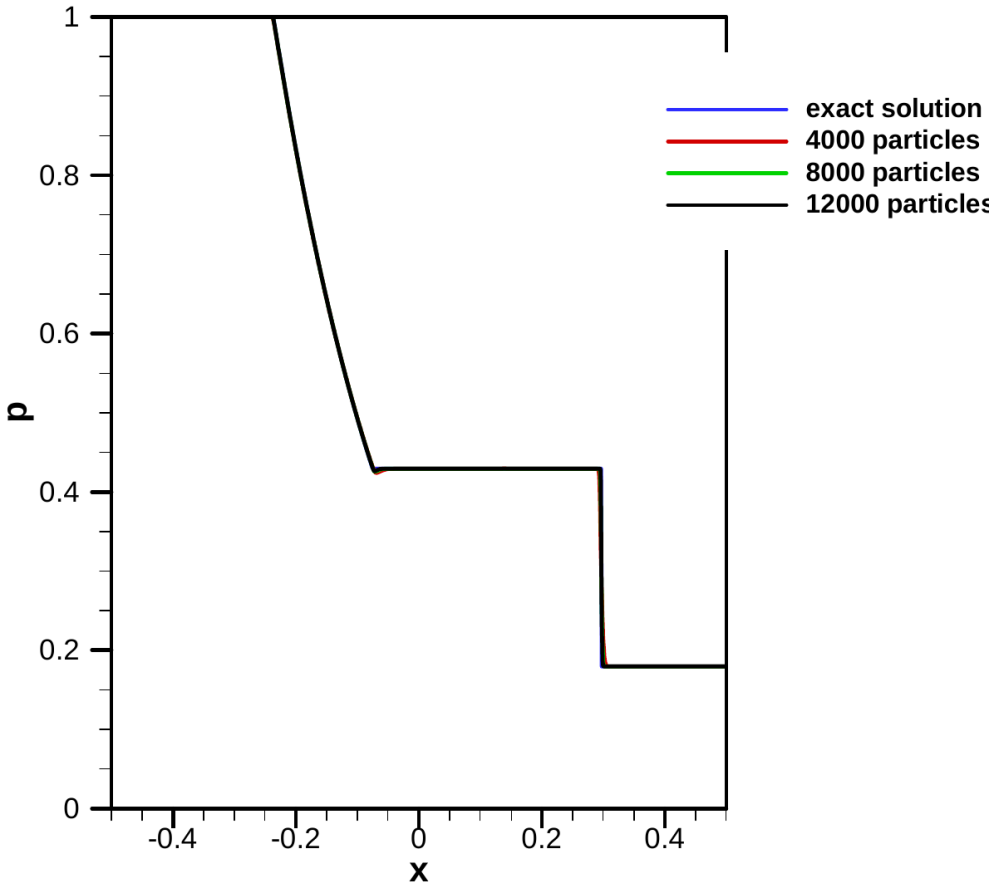
5 Figure Energy profiles for the shock tube problem obtained by G. R. Liu (2003)

as shown in figure 10 and figure 11 .

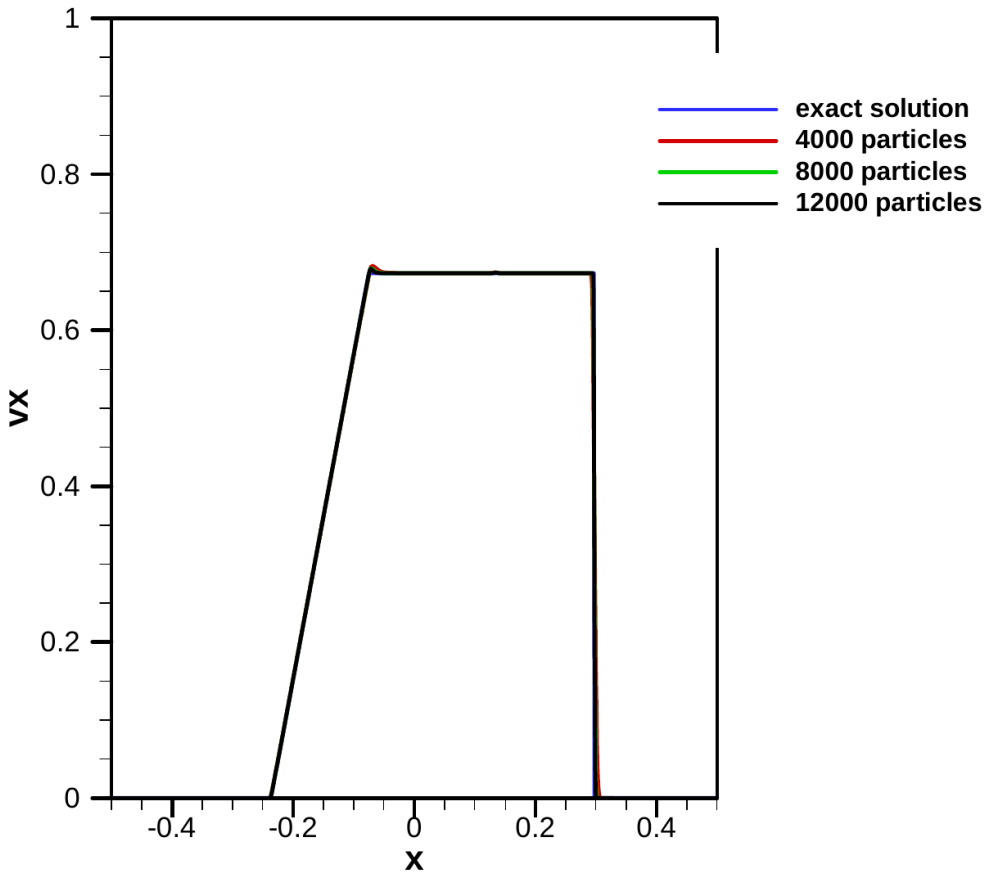
As is observed, using 12000 particles gives less the 2 % of error, it is a sutable solution for most engineering applications.



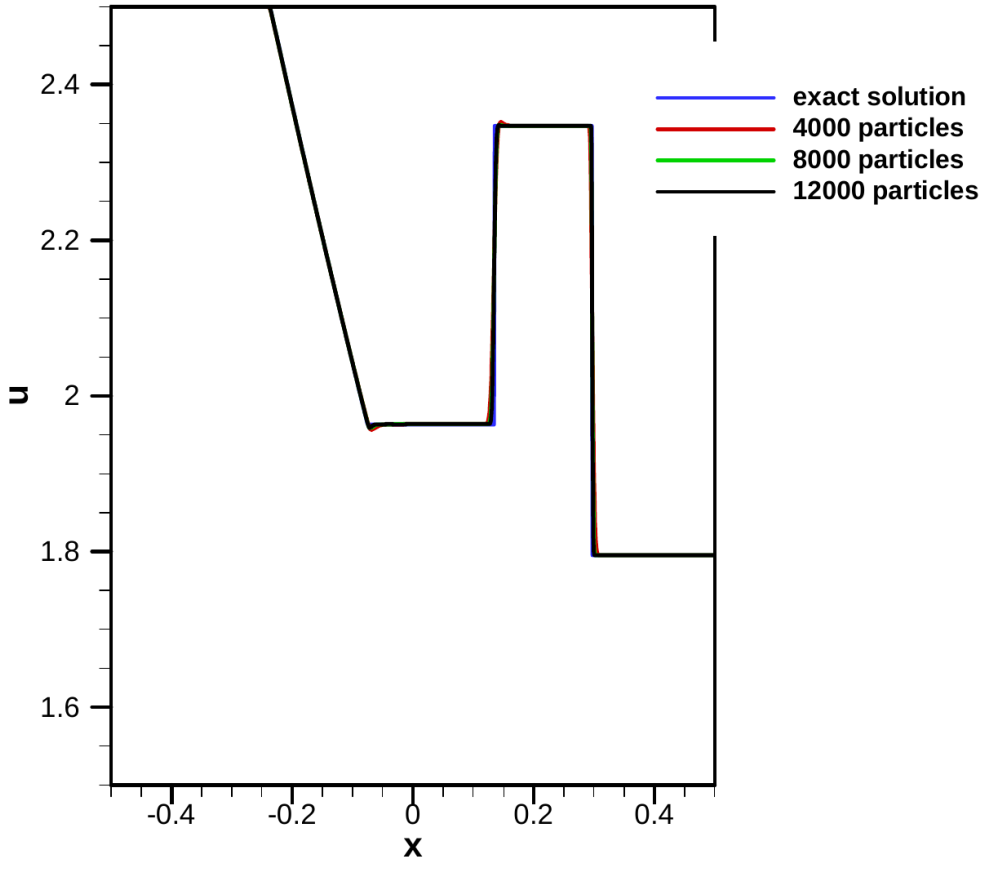
6 Figure Density profiles for the shock tube problem obtained by us



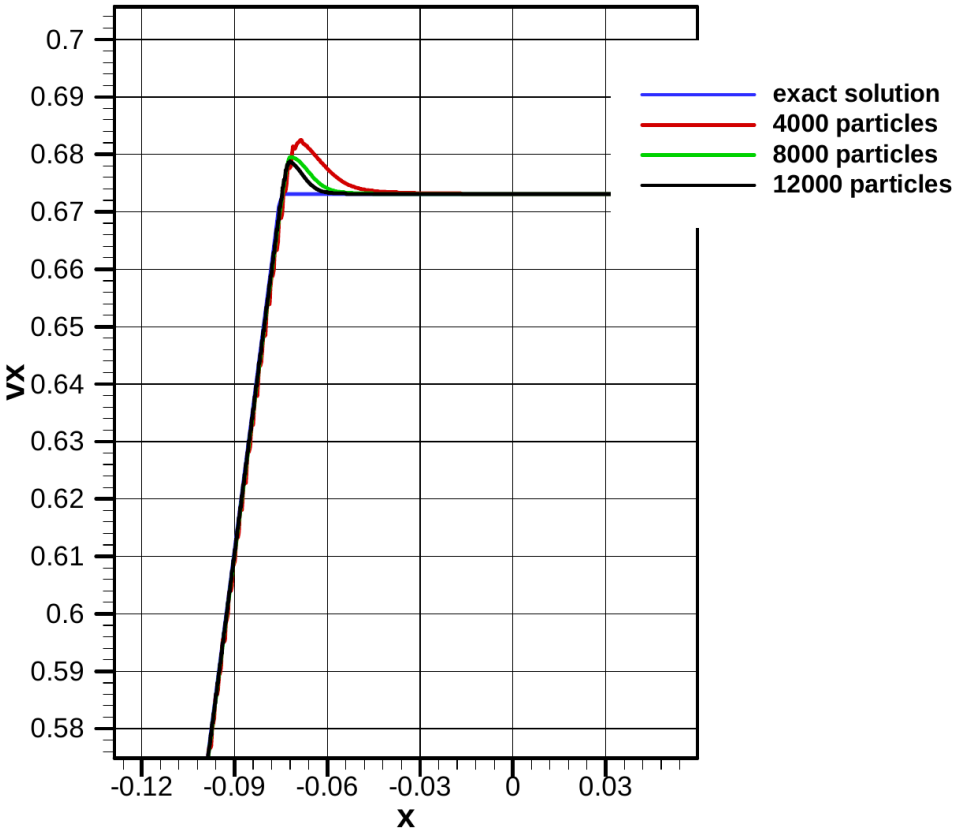
7 Figure Pressure profiles for the shock tube problem obtained by us



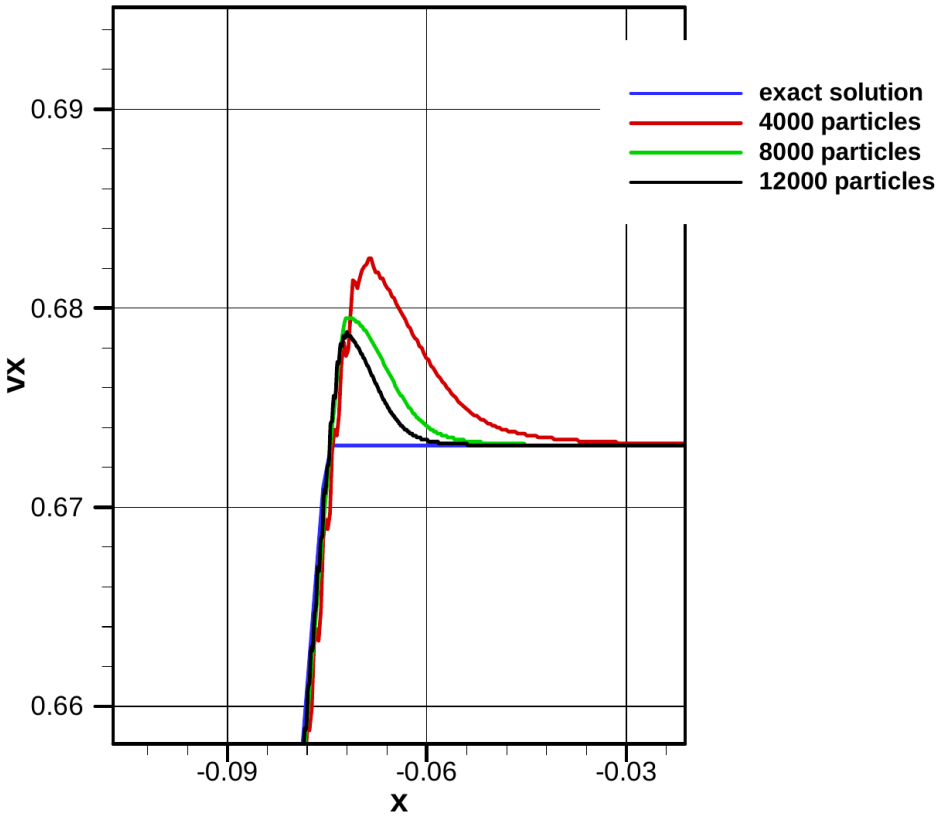
8 Figure Velocity profiles for the shock tube problem obtained by us



9 Figure Energy profiles for the shock tube problem obtained by us



10 Figure Velocity oscillations in shock region



11 Figure Velocity oscillations in shock region

6. Study of the planar stable detonation wave propagation

6.1. Problem parameters

Here, interest is focused on self-sustained Chapman-Jouguet (CJ) detonation waves [40] because of simplicity of this task.

As discussed in [1] in the inviscid limit, the activation energy plays a large role in determining the stability of the system. Moreover, the rate constant, K, merely introduces a length scale, the half-reaction length, $L_{1/2}$, (the distance between the inviscid shock and the location at which $\beta = 1/2$). In the inviscid case, linear stability analysis by Lee and Stewart [41] revealed that for $E < 25.26$, the steady ZND wave is linearly stable and is otherwise linearly unstable. Henrick et al. [1] numerically found the stability limit, for the inviscid case, at $E_a = 25.265 \pm 0.005$.

We present several calculations for the development of a detonation wave from an initially prescribed steady Chapman–Jouguet solution.

We shall try to obtain solutions for a stable detonation. We fix the specific heat ratio at $\gamma = 1.2$ and the heat release at $Q = 50$, and the activation energy $E_a = 25$ similar to that done in [1].

The steady CJ speed for this detonation is $D_{CJ} = 6.80947463$.

The equations have been scaled in such a fashion that the ambient density and pressure are $\rho_0 = 1$ and $p_0 = 1$, the half-reaction zone length, $L_{1/2}$, is unity.

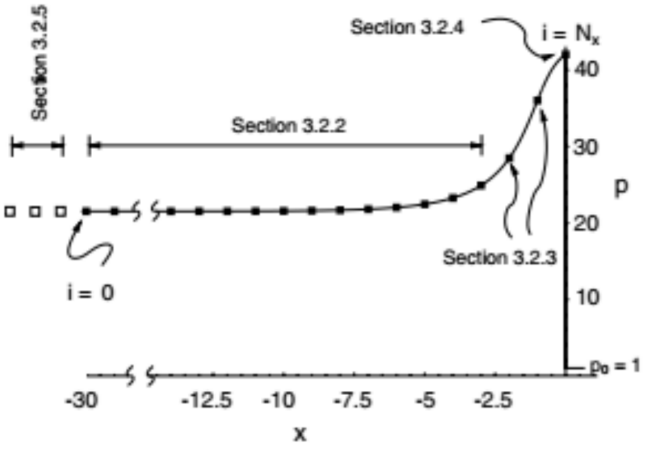
Standard approach requires one to vary k from case to case in order to maintain $L_{1/2} = 1$; for $E = 25$, one has $K = 35.955584760859722$, where the high precision is needed to guarantee the high precision of the results.

For this case, the steady solution is stable, and thus it is the exact solution for all time. This can also serve as a test problem for verification of the numerical scheme.

Our search for exact solution for this task however did not give result. Therefore we can compare our solution only by detonation speed.

In [1] given approximate figure of pressure distribution of detonation wave.

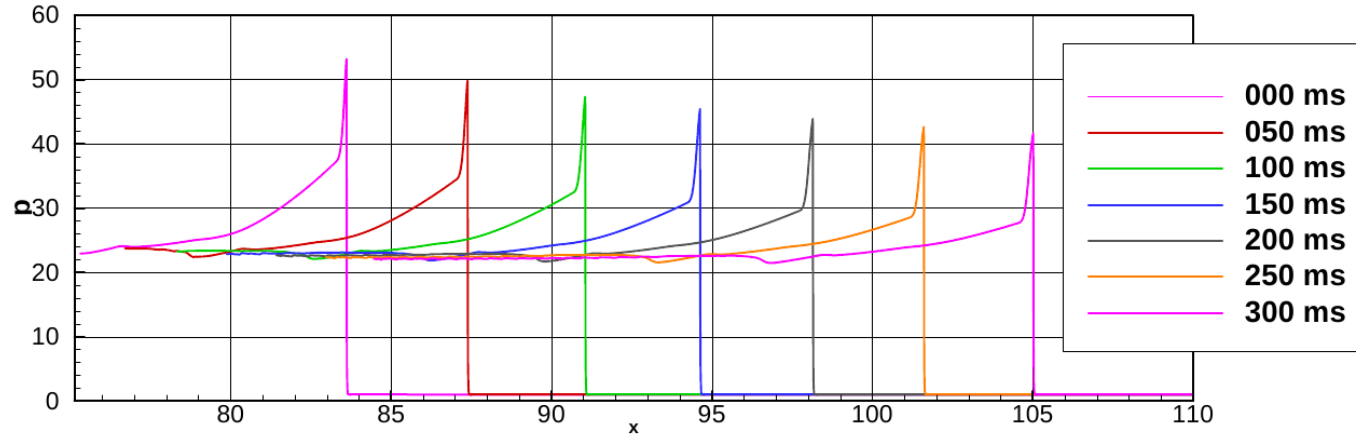
The solution for initiation of solution was obtained using ExactPack program [36]. However, the ExactPack have solution only for condensed explosives, but we had not any, therefore we used it as is.



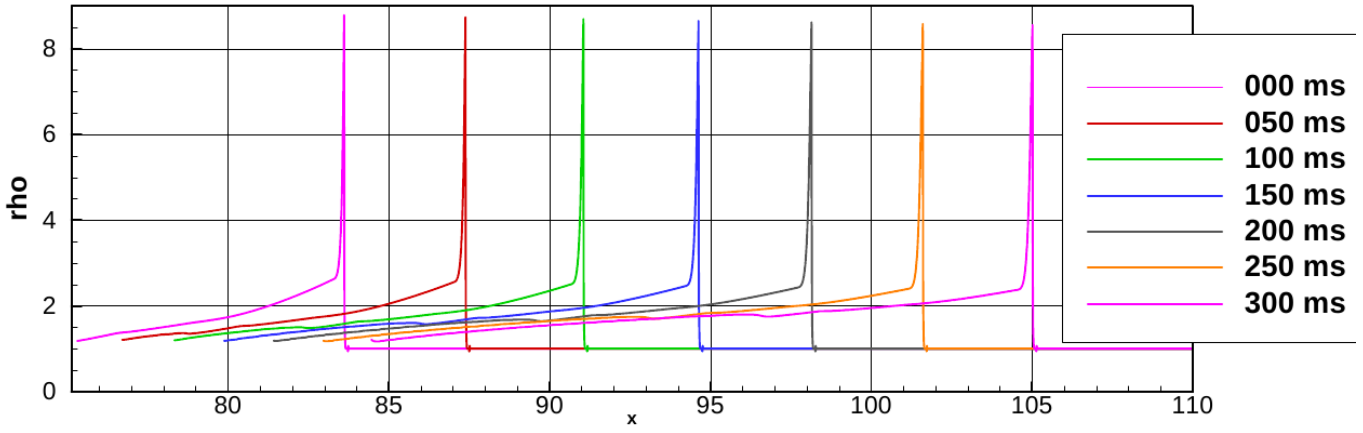
12 Figure Pressure distrubution from Henric at al. [1]

7. Results

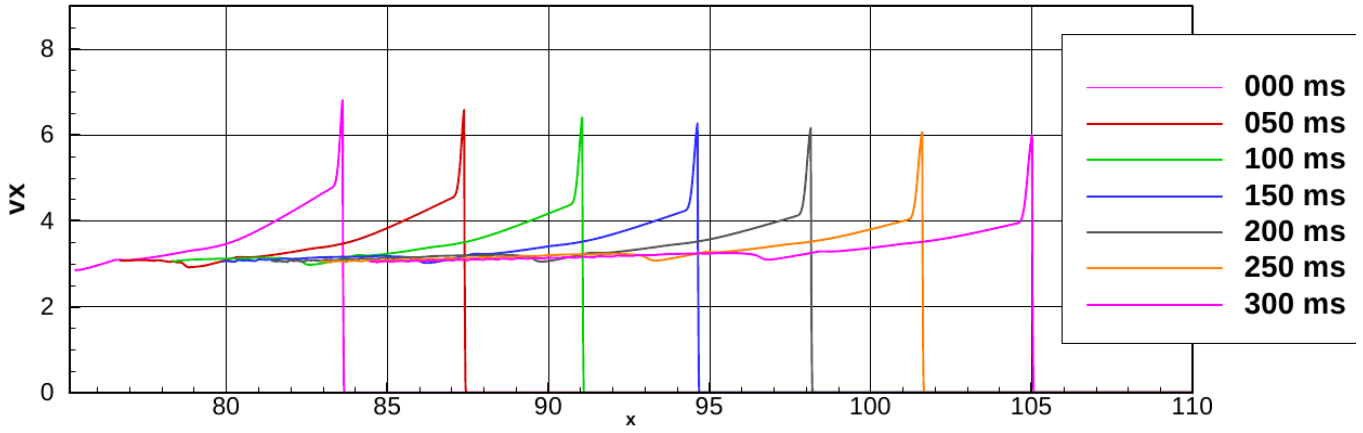
Figure 13, figure 14 , figure 15, figure 16 , figure 17 figure 18 show, respectively, the velocity, pressure, density, velocity, energy, reaction progress and reaction speed variables distribution from some time s after initiation for different time moments .



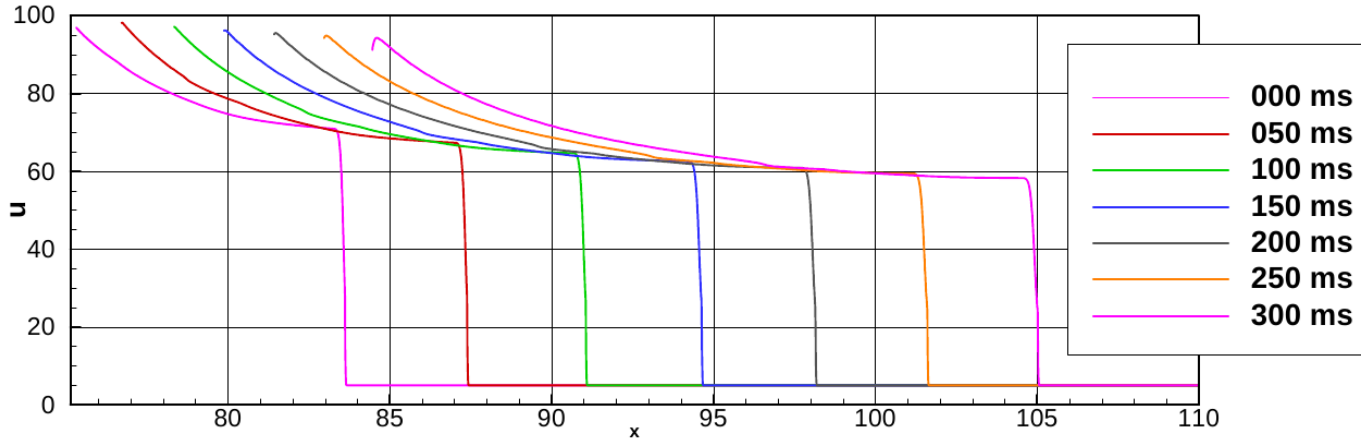
13 Figure Pressure distribution



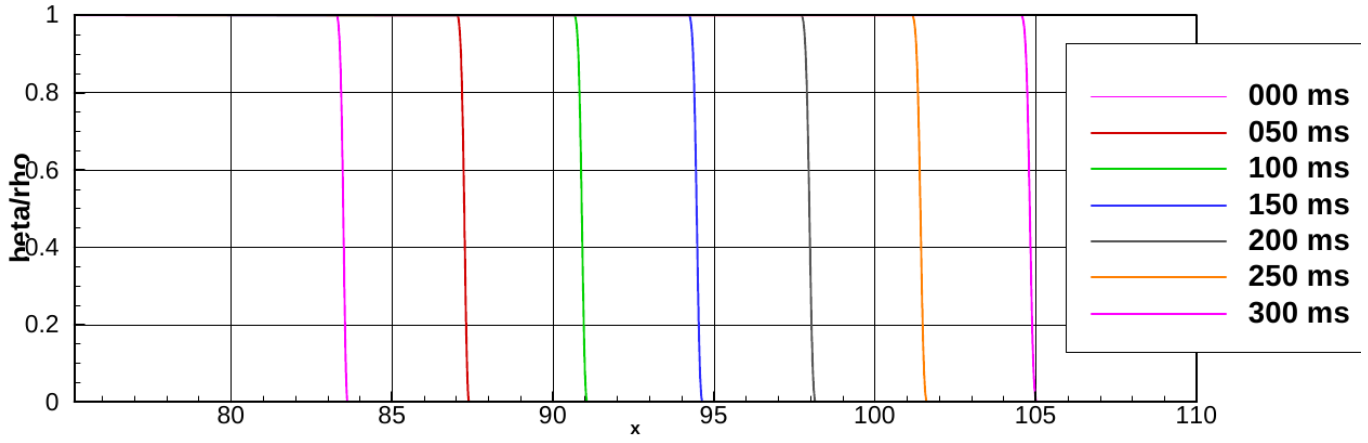
14 Figure Density distribution



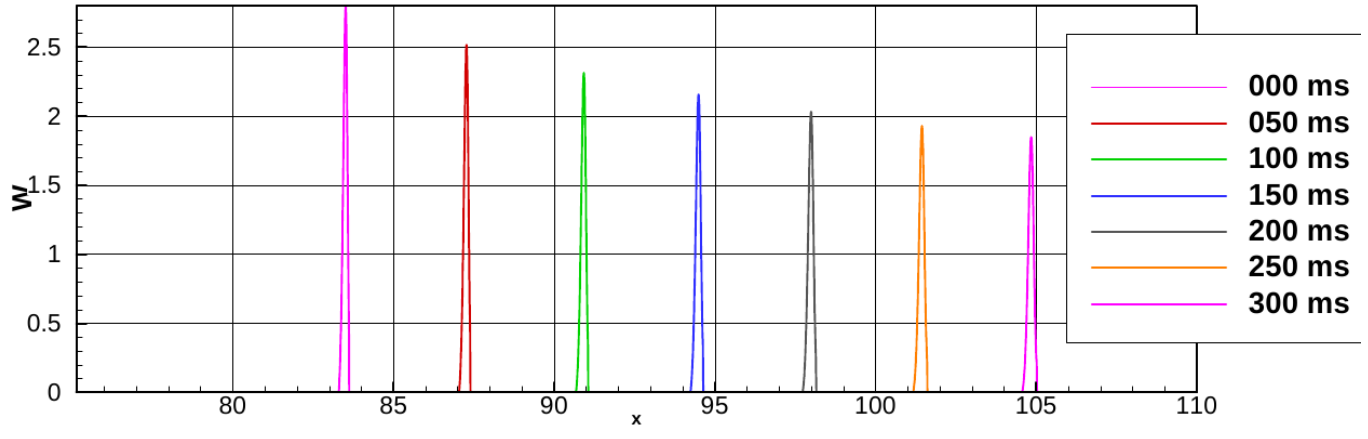
15 Figure Velocity distribution



16 Figure Energy distribution



17 Figure Reaction progress variable distribution



18 Figure Reaction speed distribution

8. Conclusion

8.1. Summary

We developed mathematical method for gas mixture detonation modeling and performed calculations of two test-cases.

Calculations were performed on Lithuanian Energy Institute computational cluster.

As we can see, modified G. R. Liu code gives good results resolving compressible gas flow in shock-tube.

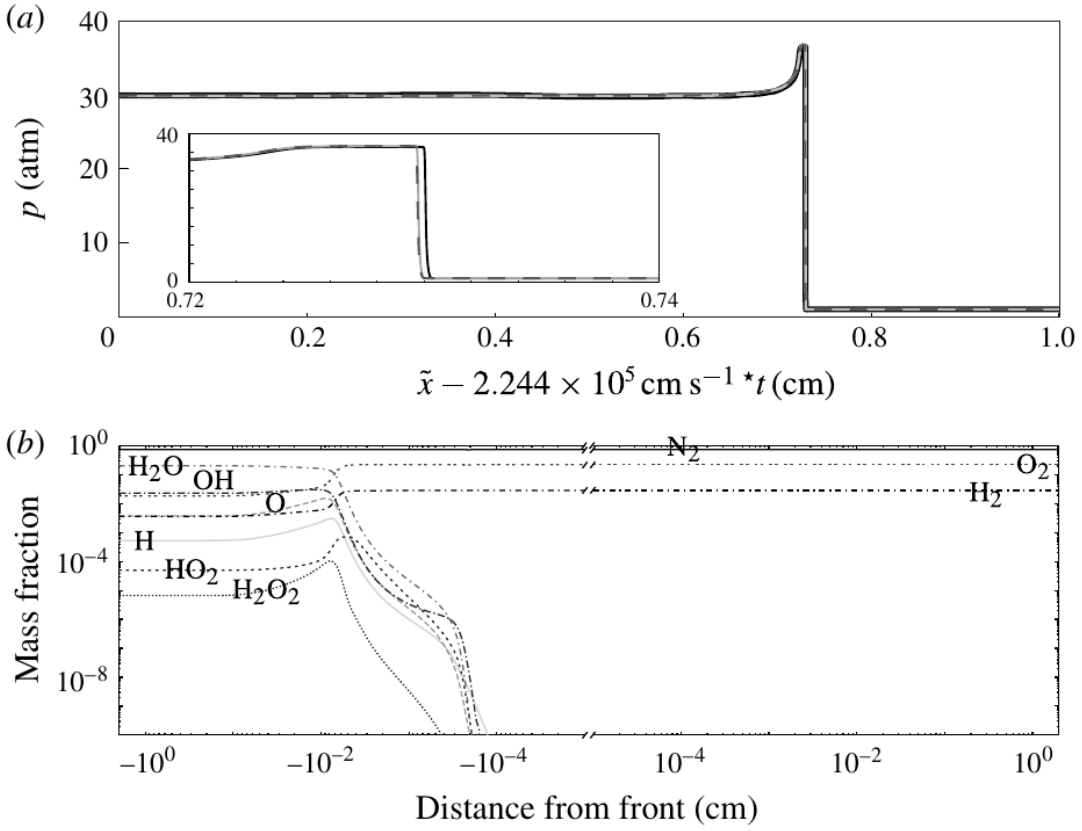
We were strictly limited in time, therefore we have not performed calculations until stable solution has been got, because we spent about three months to get results we presented in this work.

Performed calculations confirm that our model is right, although some improvements still awaiting.

We get detonation speed about 6 in the meantime desired speed is 6.8. Pressure is similar to given in [1].

8.2. Future work

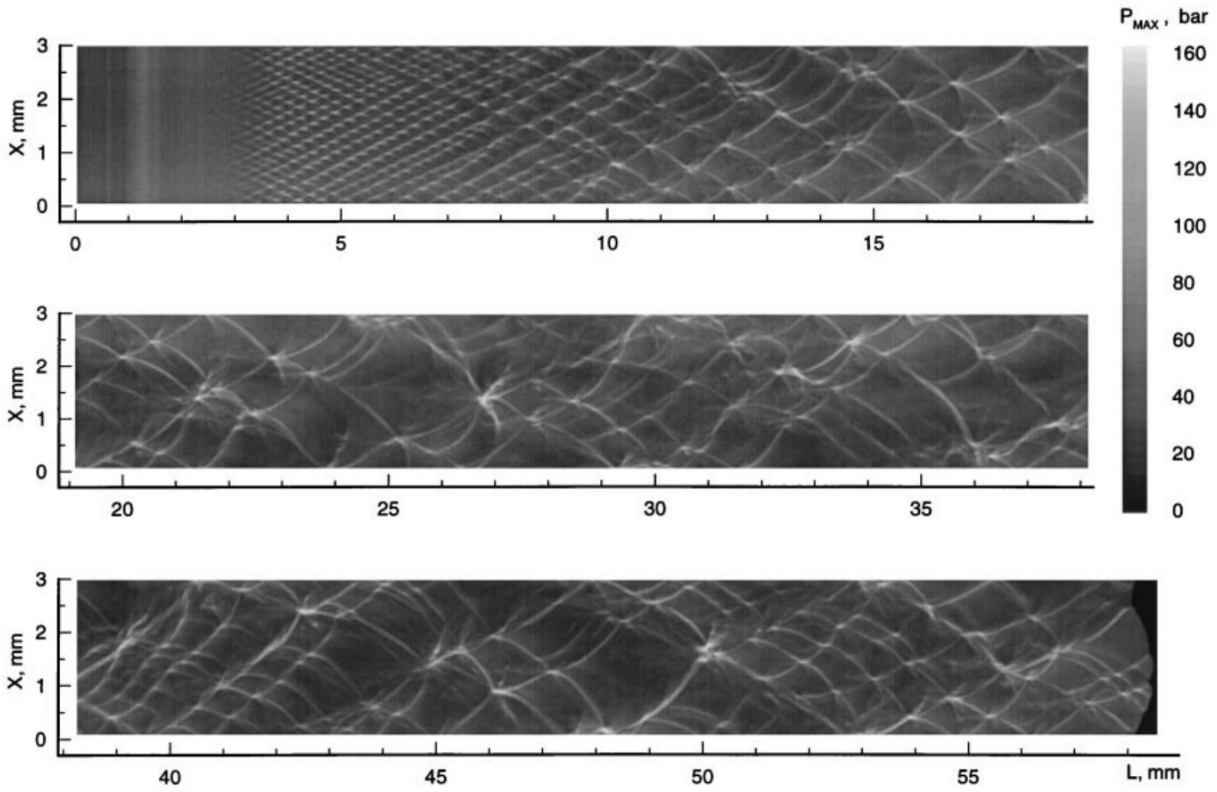
Romick, C. M.; Aslam, T. D. and Powers, J. M. in 2015 published work concerning air-hydrogen mixture detonation [42].



19 Figure (a) Several time shots of the spatial pressure profile (solid black line, 10×10^{-6} s; solid light grey line, 35×10^{-6} s; and dashed grey line, 60×10^{-6}s) and (b) typical spatial profile of mass fractions at a $up = 1.500 \times 10^5$ cm s^{-1} [42] .

Application of our approach to such task have a big practical interest. To perform such task we need to add to our model additional equations for species transportation and detailed chemical cinetics.

We are going to generalize the code for two and tree dimentional cases. Also we intend to study unstable regimes of detonation.



20 Figure Cellular detonation history presented by maximum pressure contours $E_a/RT_* = 7.4$ [2].

It will be very interesting to perform numerical simulations of cellular d etonations [2] via SPH method.

So there is a wide field for application of our method as well as for its improvement.

References

[1] A. K. Henrick, T. D. Aslam, J. M. Powers, Simulations of pulsating one-dimensional detonations with true fifth order accuracy, Journal of Computational Physics **213**, 311--329 (2006).

[2] V. N. Gamezo, D. Desbordes, E. S. Oran, Formation and evolution of two-dimensional cellular detonations, Combustion and Flame **116**, 154--165 (1999).

[3] J. B. Zel'dovich, *Detonation theory* (Gos. Tech. Teor. Izdat., Moscow, 1955).

[4] C. M. Romick, *On the effect of diffusion on gaseous detonation*, Ph.D. thesis, University of Notre Dame (2015).

[5] J. Haßlberger, *Numerical Simulation of Deflagration-to-Detonation Transition on Industry Scale*, Ph.D. thesis, Technische Universität München, Institut für Energietechnik (2017).

[6] H. D. Ng, *The effect of chemical reaction kinetics on the structure of gaseous detonations*, Ph.D. thesis, Department of Mechanical Engineering, McGill University, Montréal, Québec, Canada (2005).

[7] R. Deiterding, *Parallel Adaptive Simulation of Multi-dimensional Detonation Structures*, Ph.D. thesis, Brandenburgischen Technischen Universit¨at Cottbus (2003).

[8] C. McCabe, *Smoothed particle hydrodynamics on graphics processing units*, Ph.D. thesis, Manchester Metropolitan University (2016).

[9] L. B. Lucÿ, A numerical approach to the testing of the fission hypothesis, The Astronomical Journal (1977).

[10] R. A. Gingold, J. J. Monaghan, Smoothed particle hydrodynamics: theory and application to non-spherical stars, Mon. Not. R. astr. Soc. (1977).

[11] A. Chaniotis, *Remeshed smoothed particle hydrodynamics for the simulation of compressible, viscous, heat conduction, reacting & interfacial flows*, Ph.D. thesis, Swiss federal institute of thechnology Zurich (2002).

[12] A. Chaniotis, ohers, Remeshed smoothed particle hydrodynamics for the simulation of laminar chemically reactive flows, Journal of Computational Physics 191 (2003).

[13] SPHERIC, Sph software, http://spheric-sph.org/sph-projects-and-codes .

[14] Sphysics code, https://wiki.manchester.ac.uk/sphysics/index.php/Main_Page .

[15] D. J. Price, Ndspmhd code, http://users.monash.edu.au/ dprice/ndspmhd/ .

[16] D. Price, *Magnetic fields in Astrophysics*, Ph.D. thesis, Institute of Astronomy, University of Cambridge (2004).

[17] D. J. Price, Smoothed particle hydrodynamics and magnetohydrodynamics, Preprint submitted to J. Comp. Phys (2010).

[18] G. R. Liu, M. B. Liu, *Smoothed Particle Hydrodynamics. A meshfree particle method.* (World Scientific Publishing Co. Pte. Ltd., 2003).

[19] G.-R. Liu, Sph code, http://www.ase.uc.edu/ liugr/SPH.html .

[20] M. B. Liu, G. R. Liu, Z. Zong, K. Y. Lam, Computer simulation of high explosive explosion using smoothed particle hydrodynamics methodology, Computer and Fluids **32**, 305--322 (2003).

[21] M. B. Liu, G. R. Liu, K. Y. Lam, Z. Zong, Smoothed particle hydrodynamics for numerical simulation of underwater explosion, Computational Mechanics **30**, 106--118 (2003).

[22] M. B. Liu, G. R. Liu, K. Y. Lam, Z. Zong, Meshfree particle simulation of the detonation process for high explosives in shaped charge unlined cavity configurations, Shock Waves **12**, 509--520 (2003).

[23] D. Feng, M. B. Liu, H. Li, G. R. Liu, Smoothed particle hydrodynamics modeling of linear shaped charge with jet formation and penetration effects, Computers and Fluids **86**, 77--85 (2013).

[24] M. B. Liu, D. L. Feng, Z. M. Guo, Recent developments of sph in modelling explosion and impact problems, in *III International Conference on Particle-based Methods -- Fundamentals and Applications* (2013).

[25] G. R. Liu, G. Y. Wang, Q. Peng, S. De, A micro-macro coupling approach of md-sph method for reactive energetic materials, in *Shock Compression of Condensed Matter* (AIP Publishing, 2015), AIP Conf. Proc. 1793, 050006-1--050006-8.

[26] M. Omang, S. O. Christensen, S. Børve, J. Trulsen, Height of burst explosions: a comparative study of numerical and experimental results, Shock Waves **19**, 135--143 (2009).

[27] G. Wang, G. Liu, Q. Peng, S. De, D. Feng, M. Liu, A 3d smoothed particle hydrodynamics method with reactive flow model for the simulation of anfo, Propellants Explos. Pyrotech. **40**, 566 -- 575 (2015).

[28] G. Yang, R. Liu, D. Hu, X. Han, Simulation of one dimension shock initiation of condensed explosive by sph method, Engineering Computations **Vol. 33**(Issue: 2), 528--542 (2016).

[29] G. Wang, G. Liu, Q. Peng, S. De, A sph implementation with ignition and growth and afterburning models for aluminized explosives, International Journal of Computational Methods **14**(2) (2017).

[30] D. B. M., *LLNL Explosive Handbook, UCRL-52997*, Technical report (1981).

[31] F. A. Williams, *Combustion Theory;The Fundamental Theory of Chemically Reacting Flow Systems* (Benjamin/Cummings Publishing Company, Menlo Park, 1985).

[32] J. John D. Anderson, *Computational Fluid Dynamics. The basics with Applications* (McGraw-Hill, Inc., 1995).

[33] J. J. Monaghan, Smoothed particle hydrodynamics, Annual Review of Astronomical and Astrophysics **30**, 543--574 (1992).

[34] J. J. Monaghan, Smoothed particle hydrodynamics, school of Mathematical Sciences, Monash University, Australia (2005).

[35] L. Hernquist, N. Katz, Treesph: A unification of sph with the hierarchical tree method, The Astrophysical Journal Supplement Series **70**, 419--446 (1989).

[36] Singleton, R. J. Israel, D. M. Doebling, S. W. Woods, C. N. Kaul, A. Walter, J. W. J. Rogers, M. Lloyd, *LA-UR-16-23260*, Technical report, Los Alamos National Laboratory USA (2017-08-25).

[37] M. Liu, G. Liu, K. Lam, Investigations into water mitigation using a meshless particle method, Shock Waves **12**, 181--195 (2002).

[38] J. J. Monaghan, Silating free surface flows with sph, Journal of Computational Physics **110**, 399--406 (1994).

[39] G. A. Sod, A survey of several finite difference methods for systems of nonlinear hyperbolic conservation laws, J. Comput. Phys. (27), 1--31 (1978).

[40] W. Fickett, W. Davis, *Detonation* (University of California Press, Berkeley, CA, 1979).

[41] H. I. Lee, D. S. Stewart, Calculation of linear detonation instability: one-dimensional instability of plane detonation, Journal of Fluid Mechanics **216**, 103--132 (l990).

[42] C. M. Romick, T. D. Aslam, J. M. Powers, Verified and validated calculation of unsteady-dynamics of viscous hydrogen–air detonations, J. Fluid Mech. (2015), vol. 769, pp. 154–181 (2015).

Appendix Nr. 1.
Reaction rate variable transport module

```
      subroutine BETA_trasport_algor2_v01(ntotal,mass,niac,pair_i,
     &           pair_j,dwdx,vx,itype,x,beta, dbetadt,rho,u,omega)

c-------------------------------------------------------------------------
c     Subroutine to calculate the density with SPH continuiity approach.

c     ntotal : Number of particles                                  [in]
c     mass   : Particle masses                                      [in]
c     niac   : Number of interaction pairs                          [in]
c     pair_i : List of first partner of interaction pair            [in]
c     pair_j : List of second partner of interaction pair           [in]
c     dwdx   : derivation of Kernel for all interaction pairs       [in]
c     vx     : Velocities of all particles                          [in]
c     itype  : type of particles                                    [in]
c     x      : Coordinates of all particles                         [in]
c     rho    : Density                                              [in]
c     drhodt : Density change rate of each particle                [out]

      implicit none
      include 'param.f'

      integer ntotal,niac,pair_i(max_interaction),
     &        pair_j(max_interaction), itype(maxn)
      double precision mass(maxn), dwdx(3, max_interaction),
     &       vx(3,maxn), x(3,maxn), beta(maxn), dbetadt(maxn)
      double precision rho(maxn), eta(maxn),u(maxn)
      integer i,j,k,d
      double precision    vcc, dvx(3),he,h
      double precision c(maxn), p(maxn),omega(maxn)

!-------------------------------------------------------------------------
      do i = 1, ntotal
        dbetadt(i) = 0.

c     Pressure from equation of state

!       if (abs(itype(i)).eq.1) then
!             call p_gas(rho(i), u(i), p(i),c(i),beta(i))
!       else if (abs(itype(i)).eq.2) then             !
!             call p_art_water(rho(i), p(i), c(i))
!       endif


      enddo
!-------------------------------------------------------------------------
!-------------------------------------------------------------------------
```

```
!-----------------------------------------------------------------------
      do 21,k=1,niac
        i = pair_i(k)
        j = pair_j(k)
        he = 0.e0

!       else if (pa_sph.eq.2) then
          do 22,d=1,dim
            h = (beta(i)/rho(i)**2 + beta(j)/rho(j)**2)*dwdx(d,k)
            he = he + (vx(d,j) - vx(d,i))*h
 22       continue


        dbetadt(i) = dbetadt(i) + mass(j)*he
        dbetadt(j) = dbetadt(j) + mass(i)*he

 21   continue

!       dbetadt(i) = dbetadt(i)*beta(i)
!-----------------------------------------------------------------------
!-----------------------------------------------------------------------
!-----------------------------------------------------------------------
      return
      end
```

Appendix Nr. 2.
Boundary conditions module

```
      subroutine boundary_conditions_v02(x, vx, mass, rho, p, u,
     &         itype, hsml, beta, omega , ntotal, itimestep,
     &         x_start1old,x_start2old,x_start3old,time  )
!-------------------------------------------------------------------------
!     This subroutine is used to generate initial data for the
!     1 d noh shock tube problem
!     x--- coordinates of particles                                  [out]
!     vx--- velocities of particles                                  [out]
!     mass--- mass of particles                                      [out]
!     rho--- dnesities of particles                                  [out]
!     p--- pressure  of particles                                    [out]
!     u--- internal energy of particles                              [out]
!     itype--- types of particles                                    [out]
!          =1   ideal gas
!     hsml--- smoothing lengths of particles                         [out]
!     ntotal--- total particle number                                [out]

      implicit none
      include 'param.f'

      integer itype(maxn),itype2(maxn), ntotal, ntotal2
      integer itype5(maxn)
      integer itimestep
      integer n_detonation,N_0,daugiklis
      double precision x(3, maxn), vx(3, maxn), mass(maxn),
     &      rho(maxn), p(maxn), u(maxn), hsml(maxn), beta(maxn)
      integer i, d, j, kkk, kk_det_new , jkjk
       double precision space_x_det(120000),space_x(120000)
       double precision x2(3, maxn), vx2(3, maxn), mass2(maxn),
     &      rho2(maxn), p2(maxn), u2(maxn), hsml2(maxn), beta2(maxn)

       double precision x5(3, maxn), vx5(3, maxn), mass5(maxn),
     &      rho5(maxn), p5(maxn), u5(maxn), hsml5(maxn), beta5(maxn)

!       double precision omega(maxn)
      double precision gamma, dt, dx_omega
      double precision  temper(maxn),tabs(maxn),omega(maxn)
      double precision  temper2(maxn),tabs2(maxn),omega2(maxn)
      double precision  temper5(maxn),tabs5(maxn),omega5(maxn)
      integer i_n1,i2,i_choosen,ntotal0
      double precision  ddx
      double precision mass_particle,mass_particle2, C_1
      double precision p0
      double precision rho0
      double precision vx0
      double precision RR,space_x_det_1,x_plus1
```

```
      integer ntotal8,i5,n5
      double precision x_start1,x_start2,x_start3,x_begin

      double precision x_start1old,x_start2old,x_start3old,time

      double precision E_0, K_mast

      integer i_start1,i_start2,i_start3

      logical front_start_found
      logical front_end_found

       character*10 sss

!*******************************************************************************
!*******************************************************************************
        gamma = 1.2
        E_0 = 25.
        K_mast = 35.955584760859722
        RR = 8.3144598
        DO 801,i=1,ntotal

          temper(i) = p(i)/rho(i) + 0.0000000000000000001
          tabs(i) = temper(i) * 293. * gamma / RR
       omega(i)=K_mast*((1.e0-(beta(i)/rho(i)))**0.9)*
     &        exp((-1.)*E_0/(temper(i)))
!       omega(i) = omega(i)/rho(i)
        !   if(omega(i).LT.0.)omega(i) = 0.0000000000000000
 801    continue

      !*******************************************************************************
!*******************************************************************************
      front_start_found = .false.
      front_end_found   = .false.

      i_start1 = 0
      i_start2 = 0

      x_start1 = 0.
      x_start2 = 0.

      x_begin = x(1,2) + 3.

      do 201,i=ntotal-100,20,-1

       if(.NOT.front_start_found)then

         if((omega(i).GT.0.0001).AND.
     &      (omega(i-1).GT.omega(i)).AND.
     &      (omega(i-2).GT.omega(i-1))    )then
```

```
              front_start_found = .true.
              i_start1 = i
              x_start1 = x(1,i)

            endif
          endif

           if(front_start_found.AND.
     &       (.NOT.front_end_found).AND.
     &       (x(1,i).GT.x_begin)          )then

            if((omega(i).LT.0.00001).AND.
     &          (omega(i+1).GT.omega(i)).AND.
     &          (omega(i+2).GT.omega(i+1))    )then

              front_end_found = .true.
              i_start2 = i
              x_start2 = x(1,i)

              endif
            endif

 201      continue
!*******************************************************************************
          dt = 0.000001

        if ((itimestep.EQ.1).or.(mod(itimestep,print_step).eq.0)) then

        write(*,*)""
        write(*,*)"␣determination␣of␣detonation␣front␣"
        write(*,*)""
        write(*,*)"detonation␣front␣start␣x␣=␣",x_start1,"␣i␣=␣",i_start1
        write(*,*)"detonation␣front␣end␣␣␣x␣=␣",x_start2,"␣i␣=␣",i_start2
        write(*,*)""

          dx_omega = x_start1 - x_start2

         write(825,*)time , x_start1, dx_omega, x_start3old - dx_omega

           write(*,*)'time␣,␣x_start1,␣dx_omega,␣x_start3old␣-␣dx_omega'
           write(*,*)time , x_start1, dx_omega, x_start3old - dx_omega

        endif

        x_start1old =  x_start1
        x_start2old =  x_start2
        x_start3old =  dx_omega
!*******************************************************************************
```

```
            do 301,i=1,i_start2

                beta(i) = rho(i)

  301       continue
!*******************************************************************************

          daugiklis = 2
          mass_particle2 = (0.75/260.)*DFLOAT(daugiklis)

           N_0 = 5
           C_1 = 0.25

           rho0 = 1.0e0
           p0 = 1.0e0
           vx0 = 0.0e0
           space_x_det_1 = mass_particle2/rho0


!          do 40,i = 6000,ntotal

!  40      continue

!       p = (gamma-1) * rho * (u)
          RR = 8.3144598
          gamma = 1.2

          do 41,i=i_start1+20,ntotal

          x(1,i) = x(1,i-1) + space_x_det_1
          vx(1,i) = vx0
          mass(i) = mass_particle2
          rho(i)  = rho0
          p(i) = p0
          u(i) = p(i) / ((gamma-1.) * rho(i))
          beta(i) = 0.e0
          temper(i) = p(i)/rho(i) + 0.0000000000000000001
          tabs(i) = temper(i) * 293. * gamma /  RR
          omega(i) = 0.
          hsml(i) = C_1 * (DFLOAT(N_0)*mass(i) )/rho(i)
          itype(i) = 1

 41       continue


!*******************************************************************************
      return
      end
```

Appendix Nr. 3.
Solution Initiation module

```
      subroutine detonation_ini47(x, vx, mass, rho, p, u,
     &                        itype, hsml, ntotal, beta)
!-----------------------------------------------------------------------
!     This subroutine is used to generate initial data for the
!     1 d noh shock tube problem
!     x--- coordinates of particles                                [out]
!     vx--- velocities of particles                                [out]
!     mass--- mass of particles                                    [out]
!     rho--- dnesities of particles                                [out]
!     p--- pressure  of particles                                  [out]
!     u--- internal energy of particles                            [out]
!     itype--- types of particles                                  [out]
!          =1   ideal gas
!     hsml--- smoothing lengths of particles                       [out]
!     ntotal--- total particle number                              [out]

      implicit none
      include 'param.f'

      integer itype(maxn),itype2(maxn), ntotal, ntotal2
      integer itype5(maxn)
      integer n_detonation,N_0,daugiklis
      double precision x(3, maxn), vx(3, maxn), mass(maxn),
     &     rho(maxn), p(maxn), u(maxn), hsml(maxn), beta(maxn)
      integer i, d, j, kkk, kk_det_new , jkjk
       double precision space_x_det(120000),space_x(120000)
       double precision x2(3, maxn), vx2(3, maxn), mass2(maxn),
     &     rho2(maxn), p2(maxn), u2(maxn), hsml2(maxn), beta2(maxn)

       double precision x5(3, maxn), vx5(3, maxn), mass5(maxn),
     &     rho5(maxn), p5(maxn), u5(maxn), hsml5(maxn), beta5(maxn)


       double precision gamma
       double precision  temper(maxn),tabs(maxn),omega(maxn)
       double precision  temper2(maxn),tabs2(maxn),omega2(maxn)
       double precision  temper5(maxn),tabs5(maxn),omega5(maxn)
       integer i_n1,i2,i_choosen,ntotal0
       double precision  ddx
       double precision mass_particle,mass_particle2, C_1
       double precision p0
       double precision rho0
       double precision vx0
       double precision RR,space_x_det_1,x_plus1
       integer ntotal8,i5,n5
       double precision x_start1,x_start2,x_start3
```

```
      integer i_start1,i_start2,i_start3

       character*10 sss

      open(10,file="solution_save_2019_09_11_300.dat")


       read(10,*)ntotal

      DO 461,i=1,ntotal

      read(10,*)x5(1,i),vx5(1,i),mass5(i),rho5(i),p5(i),u5(i),beta5(i)!,itype(i),hsml(i
     &             ,temper5(i),tabs5(i),omega5(i),hsml5(i),itype5(i)
  461    continue
      close(10)

!     i_n1 = 0
!*******************************************************************************
!*******************************************************************************
!*******************************************************************************
         x_start2 = 56.80
         x_start3 = 60.05
!*******************************************************************************
        i_start1 = 0
        i_start3 = 0

        do 405,i=1,ntotal

           if((x5(1,i).GT.x_start3).AND.
     &        (x5(1,i+2).LT.(x_start3+0.03)) ) then
              i_start3 = i
           endif


  405    continue
!            x_start1 = x5(1,i_start1)
             x_start3 = x5(1,i_start3)
!*******************************************************************************
!         write(*,*)''
!          write(*,*)' i_start1 = ',i_start1
!          write(*,*)' x_start1 = ',x5(1,i_start1)
         write(*,*)''
         write(*,*)' i_start3 = ',i_start3
         write(*,*)' x_start3 = ',x5(1,i_start3)
         write(*,*)''


!*******************************************************************************
!*******************************************************************************
!*******************************************************************************
```

```
      do i=1,ntotal
         if(x5(1,i).LT.x_start2)then
                beta5(i)=rho5(i)
                omega5(i)=0.
         endif
!        hsml(i)=0.005
        itype(i)=1!

      enddo

!*************************************************************************
!*************************************************************************

!*************************************************************************
!*************************************************************************
          i5 = 1

          DO 2431,i=1,ntotal

          x(1,i)     = x5(1,i)
          vx(1,i)    = vx5(1,i)
          mass(i)    = mass5(i)
          rho(i)     = rho5(i)
          p(i)       = p5(i)
          u(i)       = u5(i)
          beta(i)    = beta5(i)
          temper(i)  = temper5(i)
          tabs(i)    = tabs5(i)
          omega(i)   = omega5(i)
          hsml(i)    = hsml5(i)
          itype(i)   = itype5(i)


2431       continue

!*************************************************************************

          daugiklis = 2
          mass_particle2 = (0.75/260.)*DFLOAT(daugiklis)

          N_0 = 5
          C_1 = 0.25

          rho0 = 1.0e0
          p0 = 1.0e0
          vx0 = 0.0e0
          space_x_det_1 = mass_particle2/rho0


!         do 40,i = 6000,ntotal
```

```
!  40      continue

!       p = (gamma-1) * rho * (u)
          RR = 8.3144598
          gamma = 1.2

          do 41,i= ntotal-500,ntotal

          x(1,i) = x(1,i-1) + space_x_det_1
          vx(1,i) = vx0
          mass(i) = mass_particle2
          rho(i)  = rho0
          p(i) = p0
          u(i) = p(i) / ((gamma-1.) * rho(i))
          beta(i) = 0.e0
          temper(i) = p(i)/rho(i) + 0.0000000000000000001
          tabs(i) = temper(i) * 293. * gamma /  RR
          omega(i) = 0.
          hsml(i) = C_1 * (DFLOAT(N_0)*mass(i) )/rho(i)
          itype(i) = 1

 41       continue


!*********************************************************************************
          open(722, file='profile_tcpl_00000000.dat')

          WRITE(722,*)'VARIABLES = "x"'   ! 1
          WRITE(722,*)'"vx"'              ! 2
          WRITE(722,*)'"mass"'            ! 3
          WRITE(722,*)'"rho"'             ! 4
          WRITE(722,*)'"p"'               ! 5
          WRITE(722,*)'"u"'               ! 6
          WRITE(722,*)'"beta"'            ! 7.1
          WRITE(722,*)'"beta/rho"'        ! 7.2
          WRITE(722,*)'"T"'               ! 8
          WRITE(722,*)'"Tabs"'            ! 9
          WRITE(722,*)'"W"'               ! 10
          WRITE(722,*)'"hsml"'            ! 11


          WRITE(722,*)' ZONE I=' ,ntotal , ', F=POINT'

          DO 421,i=1,ntotal

          write(722,521)x(1,i),vx(1,i),mass(i),rho(i),p(i),u(i),beta(i)!,itype(i),hsml(i)
     &              ,beta(i)/rho(i),temper(i),tabs(i),omega(i),hsml(i)
  421     continue
```

```
  521     format(14(E13.6,' '))

          close(722)

!*******************************************************************************
!*******************************************************************************
          close(11)
          close(12)
          close(14)


        end
```

Appendix Nr. 4.
Modified G. R. Liu time integration module

```
      subroutine time_integration23(x,vx,mass,rho,p,u,c,s,e,itype,
     & hsml,ntotal, maxtimestep, dt,
     &              beta  )

c-------------------------------------------------------------------------
c       x--- coordinates of particles                          [input/output]
c       vx--- velocities of particles                          [input/output]
c       mass--- mass of particles                                     [input]
c       rho--- dnesities of particles                          [input/output]
c       p--- pressure  of particles                            [input/output]
c       u--- internal energy of particles                      [input/output]
c       c--- sound velocity of particles                             [output]
c       s--- entropy of particles, not used here                     [output]
c       e--- total energy of particles                               [output]
c       itype--- types of particles                                  [input]
c            =1   ideal gas
c            =2   water
c            =3   tnt
c       hsml--- smoothing lengths of particles                 [input/output]
c       ntotal--- total particle number                              [input]
c       maxtimestep--- maximum timesteps                             [input]
c       dt--- timestep                                               [input]


      implicit none
      include 'param.f'

      integer itype(maxn), ntotal, maxtimestep
      double precision x(3, maxn), vx(3, maxn), mass(maxn),
     &       rho(maxn), p(maxn), u(maxn), c(maxn), s(maxn), e(maxn),
     &       hsml(maxn), dt,
     &       beta(maxn),dbeta(maxn),
     &       beta_min(maxn),temp_beta,
     &       temper(maxn),tabs(maxn),omega(maxn)
      integer i, j, k, itimestep, d, current_ts, nstart
      double precision  x_min(3, maxn), v_min(3, maxn), u_min(maxn),
     &       rho_min(maxn), dx(3,maxn), dvx(3, maxn), du(maxn),
     &       drho(maxn),  av(3, maxn), ds(maxn),
     &       t(maxn), tdsdt(maxn)
      double precision  time, temp_rho, temp_u
      character supp*8, zero_string
      character name_orig*43

      double precision  gamma,E_0,K_mast,QQ,RR

       double precision x_start1old,x_start2old,x_start3old
```

```
          double precision
     &  rho_max_1,rho_max_2,
     &  p_max_1,p_max_2,
     &  vx_max_1,vx_max_2,
     &
     &  d_rho_max_abs_2, d_rho_max_abs_1,
     &  d_p_max_abs_2,   d_p_max_abs_1,
     &  d_vx_max_abs_2,  d_vx_max_abs_1,
     &
     &  rho_max_vid,p_max_vid,vx_max_vid,
     &  d_rho_max_rel,d_p_max_rel,d_vx_max_rel,
     &
     &  dd_rho_max_abs, dd_rho_max_vid, dd_rho_max_rel,
     &  dd_p_max_abs,   dd_p_max_vid,   dd_p_max_rel,
     &  dd_vx_max_abs,  dd_vx_max_vid,  dd_vx_max_rel

          rho_max_2 = 0.
          p_max_2 = 0.
          vx_max_2 = 0.

!************************************************
!************************************************
      do i = 1, ntotal
        do d = 1, dim
          av(d, i) = 0.
        enddo
      enddo

      nstart=0
      time=0.
      current_ts=0

!************************************************
      open(821, file='max_p.dat')
      open(822, file='max_vx.dat')
      open(823, file='max_rho.dat')

      open(825, file='omega_plotis.dat')
!       open(831, file='d_max_p.dat')
!       open(832, file='d_max_vx.dat')
!       open(833, file='d_max_rho.dat')

!*******************************************************************************

      do itimestep = nstart+1, nstart+maxtimestep

        current_ts=current_ts+1
        if (mod(itimestep,print_step).eq.0) then
         write(*,*)'_____________________________________________'
```

```
          write(*,*)'  current number of time step =',
     &            itimestep,'     current time=', real(time+dt)
          write(*,*)'_____________________________________________'
         endif

c      If not first time step, then update thermal energy, density and
c      velocity half a time step

          gamma = 1.2

        if (itimestep .eq. 1) then
         call boundary_conditions_v02(x, vx, mass, rho, p, u,
     &         itype, hsml, beta, omega , ntotal, itimestep,
     &         x_start1old,x_start2old,x_start3old,time  )

          endif

         if (itimestep .ne. 1) then


!************************************************

           call boundary_conditions_v02(x, vx, mass, rho, p, u,
     &         itype, hsml, beta, omega , ntotal, itimestep)

!************************************************

         do 612, i=1,ntotal

       if(detonation)then
            if((i.lt.10))then !.and.(vx(1,i).LT.0.)
!            x(1,1)=0.6
           vx(1,i)= 3.09445e0
           rho(i)= 1.833333e0
           p(i)=22.0909091e0
           beta(i)=1.*rho(i)
           u(i)= p(i)/((1.2e0-1.e0) * rho(i))
          endif


            endif
 612       continue

           do i = 1, ntotal
             u_min(i) = u(i)
             temp_u=0.
        if (dim.eq.1) temp_u=-nsym*p(i)*vx(1,i)/x(1,i)/rho(i)
             u(i) = u(i) + (dt/2.)* (du(i)+temp_u)
             if(u(i).lt.0)  u(i) = 0.
```

```
            if (.not.summation_density) then
              rho_min(i) = rho(i)
          temp_rho=0.
           if (dim.eq.1) temp_rho=-nsym*rho(i)*vx(1,i)/x(1,i)
              rho(i) = rho(i) +(dt/2.)*( drho(i)+ temp_rho)
            endif

!            if(continuity_BETA_density ) then
              beta_min(i) = beta(i)
          temp_beta=0.
          if (dim.eq.1) temp_beta=-nsym*beta(i)*vx(1,i)/x(1,i)
              beta(i) = beta(i) +(dt/2.)*( dbeta(i)+ temp_beta)
              if(beta(i)/rho(i).GT.1.e0)beta(i) = rho(i)
!            endif

            do d = 1, dim
              v_min(d, i) = vx(d, i)
              vx(d, i) = vx(d, i) + (dt/2.)*dvx(d, i)
            enddo
          enddo

        endif

        !**************************************************************************
            gamma = 1.2
            E_0 = 25.
            K_mast = 35.955584760859722
            RR = 8.3144598
            DO 801,i=1,ntotal

              temper(i) = p(i)/rho(i) + 0.0000000000000000001
              tabs(i) = temper(i) * 293. * gamma / RR
          omega(i)=K_mast*((1.e0-(beta(i)/rho(i)))**0.9)*
     &          exp((-1.)*E_0/(temper(i)))
  !       omega(i) = omega(i)/rho(i)
            !   if(omega(i).LT.0.)omega(i) = 0.0000000000000000
 801      continue

        !**************************************************************************

c---    Definition of variables out of the function vector:

          call single_step(itimestep, dt, ntotal, hsml, mass, x, vx, u, s,
     &         rho, p, t, tdsdt, dx, dvx, du, ds, drho,itype, av,
     &              beta, dbeta,omega )



        !**************************************************************************
          QQ = 50.
```

```
              do i=1,ntotal

        dbeta(i) = (1.)*dbeta(i) + omega(i)*rho(i)!*beta(i)!/rho(i)
        du(i) = du(i) + QQ * omega(i)*rho(i) !* beta(i)!/rho(i)


              enddo

!***********************************************
!***********************************************


         if (itimestep .eq. 1) then

           do i=1,ntotal
             temp_u=0.
         if (dim.eq.1) temp_u=-nsym*p(i)*vx(1,i)/x(1,i)/rho(i)
             u(i) = u(i) + (dt/2.)*(du(i) + temp_u)
             if(u(i).lt.0)  u(i) = 0.

             if (.not.summation_density ) then
         temp_rho=0.
        if (dim.eq.1) temp_rho=-nsym*rho(i)*vx(1,i)/x(1,i)
               rho(i) = rho(i) + (dt/2.)*(drho(i)+temp_rho)
             endif

 !            if(continuity_BETA_density ) then
         temp_beta=0.
         if (dim.eq.1) temp_beta=-nsym*beta(i)*vx(1,i)/x(1,i)
               beta(i) = beta(i) +(dt/2.)*( dbeta(i)+ temp_beta)
           if(beta(i)/rho(i).GT.1.000000000000)beta(i) = rho(i)
!            endif

             do d = 1, dim
               vx(d, i) = vx(d, i) + (dt/2.) * dvx(d, i) + av(d, i)
               x(d, i) = x(d, i) + dt * vx(d, i)
             enddo
           enddo

         else

           do i=1,ntotal
             temp_u=0.
        if (dim.eq.1) temp_u=-nsym*p(i)*vx(1,i)/x(1,i)/rho(i)
             u(i) = u_min(i) + dt*(du(i)+temp_u)
             if(u(i).lt.0)  u(i) = 0.

             if (.not.summation_density ) then
               temp_rho=0.
```

```
        if (dim.eq.1) temp_rho=-nsym*rho(i)*vx(1,i)/x(1,i)
              rho(i) = rho_min(i) + dt*(drho(i)+temp_rho)
            endif

!             if(continuity_BETA_density ) then
        temp_beta=0.
       if (dim.eq.1) temp_beta=-nsym*beta(i)*vx(1,i)/x(1,i)
              beta(i) = beta(i) + dt*( dbeta(i)+ temp_beta)
         if(beta(i)/rho(i).GT.1.00000000000)beta(i) = rho(i)
!             endif
         if(x(1,i).LT.56.2)then
            beta(i) = rho(i)
         endif

            do d = 1, dim
              vx(d, i) = v_min(d, i) + dt * dvx(d, i) + av(d, i)
              x(d, i) = x(d, i) + dt * vx(d, i)
            enddo



            enddo

        endif


        time = time + dt
!*******************************************************************************
        if (mod(itimestep,save_step_my).eq.0) then

            write(supp,'(i8.8)') itimestep
            name_orig='solution_save_'//supp//'.dat'
            open(15, file=name_orig)

            write(15,*)ntotal

            DO 461,i=1,ntotal

        write(15,*)x(1,i),vx(1,i),mass(i),rho(i),p(i),u(i),beta(i)!,itype(i),hsml(i)
     &             ,temper(i),tabs(i),omega(i),hsml(i),itype(i)

  461     continue

          close(15)

        endif



        if (mod(itimestep,save_step).eq.0) then
```

```
          write(supp,'(i4.4)') itimestep
          write(*,*)'supp=',supp
          call output(x, vx, mass, rho, p, u, c, itype, hsml, ntotal,
     &       itimestep)
        endif
!*************************************************************************
!*************************************************************************
        if ((itimestep.EQ.1).or.(mod(itimestep,print_step).eq.0)) then

!*************************************************************************
        rho_max_1 = 0.
        p_max_1 = 0.
        vx_max_1 = 0.

        do 507,i=1,ntotal

         if(rho(i).GT.rho_max_1)rho_max_1 = rho(i)
         if(p(i).GT.p_max_1)p_max_1 = p(i)
         if(vx(1,i).GT.vx_max_1)vx_max_1 = vx(1,i)

507     continue
!*************************************************************************
           d_rho_max_abs_1 = rho_max_1 - rho_max_2
           rho_max_vid = (rho_max_1 + rho_max_2)*0.5e0
           d_rho_max_rel = 100.*(rho_max_1 - rho_max_2)/rho_max_vid

          dd_rho_max_abs =  d_rho_max_abs_1 - d_rho_max_abs_2
          dd_rho_max_vid = (d_rho_max_abs_1 + d_rho_max_abs_2)*0.5e0

          dd_rho_max_rel = 100.*(d_rho_max_abs_1 - d_rho_max_abs_2)
     &   /dd_rho_max_vid
!*************************************************************************
          d_p_max_abs_1 = p_max_1 - p_max_2
          p_max_vid = (p_max_1 + p_max_2)*0.5e0
          d_p_max_rel = 100.*(p_max_1 - p_max_2)/p_max_vid

         dd_p_max_abs =  d_p_max_abs_1 - d_p_max_abs_2
         dd_p_max_vid = (d_p_max_abs_1 + d_p_max_abs_2)*0.5e0

        dd_p_max_rel = 100.*(d_p_max_abs_1 - d_p_max_abs_2)/dd_p_max_vid
!*************************************************************************
         d_vx_max_abs_1 = vx_max_1 - vx_max_2
         vx_max_vid = (vx_max_1 + vx_max_2)*0.5e0
          d_vx_max_rel = 100.*(vx_max_1 - vx_max_2)/vx_max_vid

         dd_vx_max_abs =  d_vx_max_abs_1 - d_vx_max_abs_2
         dd_vx_max_vid = (d_vx_max_abs_1 + d_vx_max_abs_2)*0.5e0

          dd_vx_max_rel = 100.*(d_vx_max_abs_1 - d_vx_max_abs_2)
     &  /dd_vx_max_vid
```

```
!*****************************************************************************

          rho_max_2 = rho_max_1
          p_max_2 = p_max_1
          vx_max_2 = vx_max_1

         d_rho_max_abs_2 = d_rho_max_abs_1
         d_p_max_abs_2   = d_p_max_abs_1
         d_vx_max_abs_2  = d_vx_max_abs_1

!*****************************************************************************
          write(*,*)'-------------------------------------------------'
          write(*,*)'--------   convergence   ----------------------'
          write(*,*)' density:'
          write(*,*)'      rho_max_1:',rho_max_1
          write(*,*)' d_rho_max_abs_1:',    d_rho_max_abs_1
          write(*,*)' d_rho_max_rel (%):', d_rho_max_rel
          write(*,*)'      dd_rho_max_abs:',     dd_rho_max_abs
          write(*,*)'      dd_rho_max_rel (%):',dd_rho_max_rel
          write(*,*)''
          write(*,*)' pressure:'
          write(*,*)'      p_max_1:',p_max_1
          write(*,*)' d_p_max_abs_1:',    d_p_max_abs_1
          write(*,*)' d_p_max_rel (%):', d_p_max_rel
          write(*,*)'      dd_p_max_abs:',     dd_p_max_abs
          write(*,*)'      dd_p_max_rel (%):',dd_p_max_rel
          write(*,*)''
          write(*,*)' detonation speed:'
          write(*,*)'      vx_max_1:',vx_max_1
          write(*,*)' d_vx_max_abs_1:',    d_vx_max_abs_1
          write(*,*)' d_vx_max_rel (%):', d_vx_max_rel
          write(*,*)'      dd_vx_max_abs:',     dd_vx_max_abs
          write(*,*)'      dd_vx_max_rel (%):',dd_vx_max_rel
          write(*,*)'-------------------------------------------------'
          write(*,*)'*************************************************'



!************************************************
!       open(821, file='max_p.dat')
!       open(822, file='max_vx.dat')
!       open(823, file='max_rho.dat')
!************************************************
            write(821,840)time,p_max_1,d_p_max_abs_1,d_p_max_rel
            write(822,840)time,vx_max_1,d_vx_max_abs_1,d_vx_max_rel
            write(823,840)time,rho_max_1,d_rho_max_abs_1,d_rho_max_rel

  840     format(4(E14.5,'  '))
!*****************************************************************************
```

```
            write(*,*)
          write(*,101)'x','velocity', 'dvx'
          write(*,100)x(1,moni_particle), vx(1,moni_particle),
     &                dvx(1,moni_particle)
          write(supp,'(i8.8)') itimestep
          write(*,*)'supp=',supp


        write(supp,'(i8.8)') itimestep

        name_orig='profile_tcpl_'//supp//'.dat'


        open(12, file=name_orig)

        WRITE(12,*)'VARIABLES = "x"'   ! 1
        WRITE(12,*)'"vx"'              ! 2
        WRITE(12,*)'"mass"'            ! 3
        WRITE(12,*)'"rho"'             ! 4
        WRITE(12,*)'"p"'               ! 5
        WRITE(12,*)'"u"'               ! 6
        WRITE(12,*)'"beta"'            ! 7.1
        WRITE(12,*)'"beta/rho"'        ! 7.2
        WRITE(12,*)'"T"'               ! 8
        WRITE(12,*)'"Tabs"'            ! 9
        WRITE(12,*)'"W"'               ! 10
        WRITE(12,*)'"hsml"'            ! 11


        WRITE(12,*)' ZONE I=' ,ntotal-1, ', F=POINT'

        DO 421,i=2,ntotal

        write(12,521)x(1,i),vx(1,i),mass(i),rho(i),p(i),u(i),beta(i)!,itype(i),hsml(i)
     &             ,beta(i)/rho(i),temper(i),tabs(i),omega(i),hsml(i)
 421     continue

 521     format(12(E14.6,' '))
!          WRITE(11,*)' ZONE I=' ,1000, ', F=POINT'
!               call sod_shock_exact(11)
        close(12)

!*******************************************************************************
        endif

101     format(1x,3(2x,a12))
100     format(1x,3(2x,e12.6))

      enddo
```

```
      nstart=current_ts

      close(821)
      close(822)
      close(823)

      close(825)

!     close(831)
!     close(832)
!     close(833)

      end
```